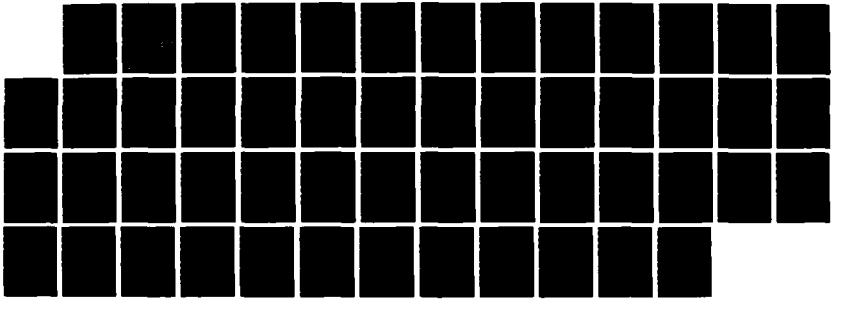
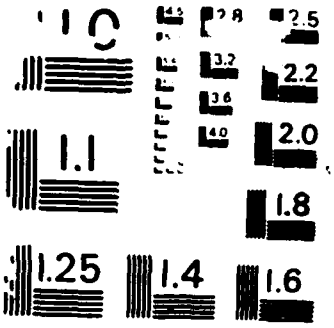
AD-A193 996 THE ORIGIN OF A BINARY-SEARCH PARADIGM REVISION(U) SRI INTERNATIONAL MENLO PARK CA Z MANNA ET AL. OCT 86 SRI-TN-351R N00039-84-C-0211 1/1

UNCLASSIFIED F/G 12/1 NL





TEST CHART

THE ORIGIN OF A BINARY-SEARCH PARADIGM

Technical Note 351R

April 1985
Revised October 1986

By: Zohar Manna
Computer Science Department
Stanford University

Richard Waldinger
Artificial Intelligence Center
Computer and Information Sciences Division

**APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITED**

This research was supported by the National Science Foundation under Grants DCR-82-14523 and DCR-85-12356, by the Defense Advanced Research Projects Agency under Contract N00039-84-C-0211, by the United States Air Force Office of Scientific Research under Contract AFOSR-85-0383, by the Office of Naval Research under Contract N00014-84-C-0706, by United States Army Research under Contract DAJA-45-84-C-0040, and by a contract from the International Business Machines Corporation.

This paper will appear in the journal *Science of Computer Programming*; a preliminary version of part of the paper appears in the proceedings of the *Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, August 1985.

ABSTRACT

In a binary-search algorithm for the computation of a numerical function, the interval in which the desired output is sought is divided in half at each iteration. The paper considers how such algorithms might be derived from their specifications by an automatic system for program synthesis. The derivation of the binary-search concept has been found to be surprisingly straightforward. The programs obtained, though reasonably simple and efficient, are quite different from those that would have been constructed by informal means.

Key Words: program synthesis; theorem proving; binary search, real square root, *lambo* function.

1. INTRODUCTION

Some of the simplest efficient algorithms for the computation of numerical functions rely on the notion of *binary search*: according to this technique, the interval in which the desired output is sought is divided in half at each iteration until it is smaller than a given tolerance.

For example, let us consider the following program for finding a real-number approximation to the square root of a nonnegative real number r. The program sets z to be

---

This research was supported by the National Science Foundation under Grants DCR-82-14523 and DCR-85-12356, by the Defense Advanced Research Projects Agency under Contract N00039-84-C-0211, by the United States Air Force Office of Scientific Research under Contract AFOSR-85-0383, by the Office of Naval Research under Contract N0001-84-C-0706, by United States Army Research under Contract DAJA-45-84-C-0040, and by a contract from the International Business Machines Corporation.

This paper will appear in the journal *Science of Computer Programming*. A preliminary version of part of the paper appears in the proceedings of the *Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, Calif., August 1985.

within a given positive tolerance ϵ less than $\sqrt{r}$.

$z \leftarrow 0;$
$v \leftarrow max(r, 1);$
while $\epsilon \leq v$ *do* $v \leftarrow v/2;$
if $[z + v]^2 \leq r$ *then* $z \leftarrow z + v;$
return(z)

This is a classical square-root program based on that of Wensley [59]. The program establishes and maintains the loop invariant that z is within v less than $\sqrt{r}$, i.e., that $\sqrt{r}$ belongs to the half-open interval $[z, z + v)$. At each iteration, the program divides this interval in half and tests whether $\sqrt{r}$ is in the right or left half, adjusting z and v accordingly, until v is smaller than the given tolerance ϵ. The program is reasonably efficient; it terminates after $\lceil log_2(max(r, 1)/\epsilon) \rceil$ iterations.

Analogous programs provide an efficient means of computing a variety of numerical functions. It is not immediately obvious how such programs can be developed by automatic program-synthesis systems, which derive programs to meet given specifications. Some researchers (e.g., Dershowitz and Manna [77], Smith [85]) have suggested that synthesis systems be provided with several general program schemata, which could be specialized to fit particular applications. Binary search would be one of these schemata. The system would have to determine which schema, if any, is applicable to a new problem.

It may indeed be valuable to provide a synthesis system with general schemata, but this approach leaves open the question of how such schemata are discovered in the first place. To our surprise, we have found that the concept of binary search emerges quite naturally and easily in the derivations of some numerical programs and therefore does not need to be built in. The programs we have obtained in this way are simple and efficient, but bizarre in appearance and quite different from those we would have constructed by informal means.

We have derived the programs in a deductive framework (Manna and Waldinger [80]) in which the process of constructing a program is regarded as a task of proving a mathematical theorem. According to this approach, the program's specification is phrased as a theorem, the theorem is proved, and a program guaranteed to meet the specification is extracted from the proof. If the specification reflects our intentions correctly, no further verification or testing is required.

In this paper we outline our deductive framework and show the derivation of a novel real-number square-root program, emphasizing the emergence of the binary-search concept. We then show several analogous binary-search derivations, for both different problems and different specifications of the same problem.

2. DEDUCTIVE PROGRAM SYNTHESIS

In this section we present our framework briefly, using the square-root derivation as a continuing example.

We begin with an outline of the logical concepts we shall need.

LOGICAL PREREQUISITES

The system deals with

- *terms* composed (in the usual way) of constants $a, b, c, \ldots$, variables $u, v, w, \ldots$, function symbols, and the conditional term constructor *if-then-else*.
- *atoms* composed of relation (predicate) symbols, including the equality symbol $=$, applied to terms, and the truth symbols *true* and *false*.
- *sentences* composed of atoms and logical connectives.

Sentences are quantifier-free. An *expression* is a term or a sentence. An expression is said to be *ground* if it contains no variables. We sometimes use infix notation for function and relation symbols (for example, $x + a$ or $0 \prec_w y$). Certain of the symbols are declared to be *primitive*; these are the computable symbols of our programming language.

We loosely follow the terminology of Robinson [79]. We denote a substitution θ by $\{x_1 \leftarrow t_1, x_2 \leftarrow t_2, \ldots, x_n \leftarrow t_n\}$. For any expression e, the expression $e\theta$ is the result of *applying* θ *to* e, obtained by simultaneously replacing every occurrence of the variable x_i in e with the corresponding term t_i. We shall also say that $e\theta$ is an *instance* of e.

Let e, s, and t be expressions, where s and t are either both sentences or both terms. If we write e as $e[s]$, then $e[t]$ denotes the result of replacing every occurrence of s in $e[s]$ with t. Let θ be a substitution. Then $e\theta[t]$ denotes the result of replacing every occurrence of $s\theta$ in $e\theta$ with t.

Variables in sentences are given an implicit universal quantification; a sentence is true under a given interpretation if every instance of the sentence is true, or, equivalently, if every ground instance of the sentence (i.e., an instance that contains no variables) is true.

We now describe the basic notions of deductive program synthesis.

SPECIFICATIONS AND PROGRAMS

A specification is a statement of the purpose of the desired program, which does not need to indicate a method of achieving that purpose. In this paper we consider only

DEDUCTIVE TABLEAUX

The fundamental structure of our system, the deductive tableau, is a set of *rows*, each of which must contain a sentence, either an *assertion* or a *goal*; any of these rows may also contain a term, the *output entry*. An example of a tableau follows:

assertions	goals	outputs $sqrt(r, \epsilon)$
1. $0 \leq r$ *and* $0 < \epsilon$		
	2. $z^2 \leq r$ *and* *not* $[(z+\epsilon)^2 \leq r]$	z
	3. *not* $[\epsilon^2 \leq r]$	0

Here z is a variable and r and ϵ are constants.

Under a given interpretation for its constant, function, and predicate symbols, a tableau is true whenever the following condition holds:

> If all instances of each of the assertions are true,
> then some instance of at least one of the goals is true.

Equivalently, the tableau is true if some instance of at least one of the assertions is false or some instance of at least one of the goals is true. Thus, the above tableau is true if assertion 1,

$$0 \leq r \ \textit{and} \ 0 < \epsilon,$$

is false or if the instance (obtained by taking z to be 0) of goal 2

$$0^2 \leq r \ \textit{and} \ \textit{not} \ [(0+\epsilon)^2 \leq r]$$

is true (among other possibilities).

In a given theory, a tableau is said to be *valid* if it is true under any model for the theory. In the theory of real numbers, the above tableau is valid, since it is true under any model. For either assertion 1 is false, or r is nonnegative and the instance of goal 2 obtained by taking z to be $\sqrt{r}$ is true.

Under a given interpretation and for a given specification

$$f(a) \Leftarrow \text{find } z \text{ such that } \mathcal{R}[a, z] \quad \text{where } \mathcal{P}[a],$$

a goal is said to have a *suitable* output entry if, whenever an instance of the goal is true, the corresponding instance t' of the output entry will satisfy the *input-output* condition

$$\textit{if } \mathcal{P}[a] \textit{ then } \mathcal{R}[a, t'].$$

(If the goal has no explicit output entry, it is said to have a suitable output entry if, whenever an instance of the goal is true, any term t' satisfies the input-output condition.) An assertion is said to have a suitable output entry if, whenever an instance of the assertion is false, the corresponding instance t' of the output entry will satisfy the input-output condition.

For example, in the theory of real numbers, consider the square-root specification

$$\begin{aligned} &sqrt(r, \epsilon) \Leftarrow \text{find } z \text{ such that} \\ &\qquad\qquad z^2 \leq r \textit{ and not } \left[(z+\epsilon)^2 \leq r\right] \\ &\quad \text{where } 0 \leq r \textit{ and } 0 < \epsilon. \end{aligned}$$

Under any model for the theory, the output entries of the above tableau are suitable for the square-root specification. In particular, if some instance of goal 2, obtained by replacing z with a term s, is true, then s will satisfy the input-output condition,

$$\begin{aligned} &\textit{if } 0 \leq r \textit{ and } 0 < \epsilon \\ &\textit{then } s^2 \leq r \textit{ and not } \left[(s+\epsilon)^2 \leq r\right]. \end{aligned}$$

Also, if assertion 1, which has no output entry, is false, then any term s satisfies the above condition.

Under a given interpretation J and for a given specification, two tableaux $\mathcal{T}_1$ and $\mathcal{T}_2$ have the *same meaning* if

$\mathcal{T}_1$ is true under J
  if and only if
$\mathcal{T}_2$ is true under J

and

the output entries of $\mathcal{T}_1$ are suitable
  if and only if
the output entries of $\mathcal{T}_2$ are suitable.

In a given theory and for a given specification, two tableaux are *equivalent* if, under any model J for the theory, the two tableaux have the same meaning.

We shall use the following properties of a tableau (for a particular theory and a particular specification):

- *Duality Property*

Any tableau is equivalent to the one obtained by removing an assertion and adding its negation as a new goal, with the same output entry. Similarly, any tableau is equivalent

applicative (or *functional*) programs, which yield an output but alter no data structures and produce no other side effects. The specifications for these programs have the form

$$f(a) \Leftarrow \text{find } z \text{ such that } \mathcal{R}[a, z] \quad \text{where } \mathcal{P}[a].$$

In other words, the program f that we want to construct is to yield, for a given *input* a, an *output* z satisfying the *output condition* $\mathcal{R}[a, z]$, provided that the input a satisfies the *input condition* $\mathcal{P}[a]$. In other words, z is to satisfy the *input-output* condition

$$\textit{if } \mathcal{P}[a] \textit{ then } \mathcal{R}[a, z].$$

For example, suppose we want to specify the program *sqrt* to yield a real number z that is within a given tolerance ϵ less than $\sqrt{r}$, the exact square root of a given nonnegative real number r. Then we might write

$$sqrt(r, \epsilon) \Leftarrow \text{find } z \text{ such that } z^2 \leq r \textit{ and not } [(z+\epsilon)^2 \leq r] \quad \text{where } 0 \leq r \textit{ and } 0 < \epsilon.$$

In other words, we want to find an output z satisfying the output condition

$$z^2 \leq r \textit{ and not } [(z+\epsilon)^2 \leq r],$$

provided that the inputs r and ϵ satisfy the input condition

$$0 \leq r \textit{ and } 0 < \epsilon.$$

The above square-root specification is not a program and does not indicate a particular method for computing the square root; it describes the input-output behavior of many programs, employing different algorithms and perhaps producing different outputs. Of course, other specifications for a square-root program are possible.

The programs we consider are sets of expressions of the form

$$f_i(a) \Leftarrow t_i,$$

where t_i is a *primitive* term, i.e., one expressed entirely in the vocabulary of our programming language. We regard the input a as primitive. These programs can be mutually recursive; i.e., we also regard the function symbols f_i as primitive. In the usual way, such a program indicates a method for computing an output.

In a given theory, a program f is said to *satisfy* a specification of the above form if, for any input a satisfying the input condition $\mathcal{P}[a]$, the program $f(a)$ terminates and produces an output t satisfying the output condition $\mathcal{R}[a, t]$. The problem we face is to construct a program satisfying a given specification.

to the one obtained by removing a goal and adding its negation as a new assertion. Thus, we could manage with a system that has no goals or a system that has no assertions, but the distinction between assertions and goals does make derivations easier to understand.

- *Renaming Property*

Any tableau is equivalent to the one obtained by systematically renaming the variables of any row. More precisely, we may replace any of the variables of the row with new variables, making sure that all occurrences of the same variable in the row (including those in the output entry) are replaced by the same variable and that distinct variables in the row are replaced by distinct variables. In other words, the variables of a row are dummies that may be renamed freely.

- *Instance Property*

Any tableau is equivalent to the one obtained by introducing as a new row any instance of an existing row. The new row is obtained by replacing all occurrences of certain variables in the existing row (including those in the output entry) with terms. Note that the existing row is not replaced; the new one is simply added.

THE DEDUCTIVE PROCESS

Consider a particular theory and the specification

$f(a) \Leftarrow$ find z such that $\mathcal{R}[a, z]$
where $\mathcal{P}[a]$.

We form the initial tableau

assertions	goals	outputs $f(a)$
$\mathcal{P}[a]$		
	$\mathcal{R}[a, z]$	z

Here the input condition $\mathcal{P}[a]$ is the initial assertion, the output condition $\mathcal{R}[a, z]$ is the initial goal, and the output z is the goal's output entry. We regard the input a as a constant and the output z as a variable. We may also include in the initial tableau (as an assertion) any valid sentence of the theory.

Note that the output entries of this tableau are suitable. Under any model for the theory, if the initial assertion $\mathcal{P}[a]$ is false, then any output satisfies the input-output condition vacuously; and if some instance $\mathcal{R}[a, t']$ of the initial goal $\mathcal{R}[a, z]$ is true, the corresponding instance t' of the associated output entry satisfies the input-output condition. Furthermore, the valid sentences included as initial assertions cannot be false.

Example

For the specification of the real-number square-root program,

$$sqrt(r, \epsilon) \Leftarrow \text{find } z \text{ such that } z^2 \leq r \text{ and not } [(z+\epsilon)^2 \leq r]$$
$$\text{where } 0 \leq r \text{ and } 0 < \epsilon,$$

we form the initial tableau

assertions	goals	outputs $sqrt(r, \epsilon)$
1. $0 \leq r$ *and* $0 < \epsilon$		
	2. $z^2 \leq r$ *and* *not* $[(z+\epsilon)^2 \leq r]$	z

Here the inputs r and ϵ are constants and the output z is a variable. We may also include as assertions valid sentences of the theory of real numbers, such as

$u^2 = u \cdot u$		
$0 \cdot v = 0$		

where u and v are variables. ∎

In the deductive process, we attempt to show that the initial tableau is valid. For this purpose, we apply deduction rules that add new rows without changing the tableau's meaning in any model for the theory. In other words, under a given model, the tableau is true before application of the rule if and only if it is true afterwards, and the output entries are suitable beforehand if and only if they are suitable afterwards. We describe the deduction rules in the next section.

The process continues until we obtain either of the two rows

	true	t

or

false		t

where the output entry t is primitive, i.e., expressed entirely in the vocabulary of our programming language. At this point, we derive the program

$$f(a) \Leftarrow t.$$

We claim that t satisfies the given specification. For, in applying the deduction rules, we have guaranteed that the new output entries will be suitable if the earlier output entries are suitable. We have seen that the initial output entries are all suitable; therefore, the final output entry t is also suitable. This means that, under any model, if the final goal *true* is true or the final assertion *false* is false, the corresponding output entry t will satisfy the input-output condition

$$\textit{if } \mathcal{P}[a] \textit{ then } \mathcal{R}[a, t].$$

But, under any model, the truth symbols *true* and *false* are true and false, respectively, and hence t will satisfy the input-output condition. Therefore, the program $f(a) \Leftarrow t$ does satisfy the specification.

For example, from the square-root derivation we shall obtain the program

$$sqrt(r, \epsilon) \Leftarrow \begin{cases} \textit{if } max(r, 1) < \epsilon \\ \textit{then } 0 \\ \textit{else if } \left[sqrt(r, 2\epsilon) + \epsilon\right]^2 \leq r \\ \qquad \textit{then } sqrt(r, 2\epsilon) + \epsilon \\ \qquad \textit{else } sqrt(r, 2\epsilon). \end{cases}$$

(Actually we shall obtain a slightly different program.) Before we describe the deduction rules of our system, let us say a few words about this program. This will help the understanding of the ensuing derivation.

DISCUSSION OF THE PROGRAM

The program first checks whether the error tolerance ϵ is reasonably small. If ϵ is very big, that is, if $max(r, 1) < \epsilon$, then the output can safely be taken to be 0. For, because $0 \leq r$, we have

$$0^2 \leq r.$$

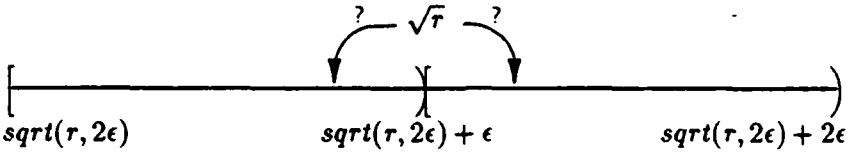
And because $max(r, 1) < \epsilon$, we have $r < \epsilon$ and $1 < \epsilon$, and hence $r < \epsilon^2$ — that is,

$$\textit{not } \left[(0 + \epsilon)^2 \leq r\right].$$

Thus, in this case, taking z to be 0 satisfies both conjuncts of the output condition

$$z^2 \leq r \textit{ and not } \left[(z + \epsilon)^2 \leq r\right].$$

If ϵ is small, that is, if $\epsilon \leq max(r, 1)$, the program finds a rougher estimate $sqrt(r, 2\epsilon)$, which is within 2ϵ less than $\sqrt{r}$, the exact square root of r. In other words, the root is within the half-open interval $\left[sqrt(r, 2\epsilon), sqrt(r, 2\epsilon) + 2\epsilon\right)$. The program then asks whether $\left[sqrt(r, 2\epsilon) + \epsilon\right]^2 \leq r$, that is, whether the root is in the right or the left half of this interval. The situation is illustrated below:



If the root is in the right half, we can increase our rough estimate by ϵ, for $sqrt(r, 2\epsilon)+\epsilon$ is then within ϵ less than the root. On the other hand, if the root is within the left half, we can leave the estimate alone, for $sqrt(r, 2\epsilon)$ is already within ϵ less than the root.

The termination of the program may seem a bit problematic, because the argument ϵ is doubled with each recursive call. However, the argument r is unchanged and recursive calls are evaluated only until $max(r, 1) < \epsilon$, so there is a uniform upper bound on these increasing arguments.

If the multiple occurrences of the recursive call $sqrt(r, 2\epsilon)$ are combined by eliminating common subexpressions, the program we obtain is reasonably efficient; it requires $\lceil log_2(max(r, 1)/\epsilon) \rceil$ recursive calls. Furthermore, the resulting program is of "linear" form and may be transformed into an iterative equivalent (Harrison and Khoshnevisan [86]).

Our final program is somewhat different from the iterative program we considered in the introduction. The iterative program divides an interval in half at each iteration; the recursive program doubles an interval with each recursive call. Division of the interval occurs implicitly as the recursive program unwinds, i.e., when the recursive calls finally yield output values. Our program may actually be superior if doubling a real number is faster than halving one.

It is possible to obtain a version of the iterative program by formal derivation from the specification within the deductive-tableau system. Although the derivation and the resulting program are more complex (the program requires two additional inputs), it was this more complex derivation we discovered first, as we were already familiar with the iterative program.

We later found the recursive program while examining the consequences of purely formal derivation steps, not because we expected them to lead to a program, but because we were looking for strategic considerations that would rule out these branches of the search space. When we examined the program initially, we suspected an error in the derivation. We had not seen programs of this form before, and we certainly would not have constructed this one by informal means.

THE TRANSFORMATION RULES

We now begin to introduce the deduction rules of our system, illustrating them with fragments from the square-root derivation. Afterwards, we shall review the entire derivation. We begin with the simplest of the rules.

The *transformation rules* replace subexpressions of an assertion, goal, or output entry with equal or equivalent expressions. For instance, with the transformation rule

$$\mathcal{P} \textit{ and } \textit{true} \rightarrow \mathcal{P},$$

we can replace the subsentence $((A \textit{ or } B) \textit{ and } \textit{true})$ with $(A \textit{ or } B)$ in the assertion

$((A \textit{ or } B) \textit{ and } \textit{true}) \textit{ or } D$		0

yielding

$(A \textit{ or } B) \textit{ or } D$		0

With the transformation rule (in the theory of integers or reals)

$$u + u \rightarrow 2u,$$

we can replace a subterm $(a + b) + (a + b)$ with the term $2(a + b)$.

We use an *associative-commutative* matching algorithm (Stickel [81]), so that the associative and commutative properties of operators can be taken into account in applying the transformation rules. Thus, we can use the above rules to replace a subsentence $(\textit{true } \textit{and } B)$ with the sentence B and the subterm $(a + b) + b$ with the term $a + 2b$.

We include a complete set of *true-false* transformation rules, such as

$$\textit{not false} \rightarrow \textit{true}$$

$$\textit{if } \mathcal{P} \textit{ then false} \rightarrow \textit{not } \mathcal{P}.$$

Repeated application of these rules can eliminate from a tableau row any occurrence of a truth symbol *true* or *false* as a proper subsentence.

The soundness of the transformation rules is evident, since each produces an expression equivalent or equal (in the theory) to the one to which it is applied.

3. CONDITIONAL FORMATION

In this section we introduce the resolution rule, which can account for the introduction of the conditional (*if-then-else*) construct into the derived program.

THE RESOLUTION RULE: GROUND VERSION

The *resolution rule* corresponds to case analysis in informal reasoning. We first

present the *ground version* of the rule, which applies to ground goals, i.e., goals with no variables. We express it in the following notation:

assertions	goals	outputs $f(a)$
	$\mathcal{F}[\mathcal{P}]$	s
	$\mathcal{G}[\mathcal{P}]$	t
	$\mathcal{F}[true]$ *and* $\mathcal{G}[false]$	*if* $\mathcal{P}$ *then* s *else* t

In other words, suppose that our tableau contains two ground goals, $\mathcal{F}$ and $\mathcal{G}$, whose output entries are s and t, respectively. Suppose further that $\mathcal{F}$ and $\mathcal{G}$ have a common subsentence $\mathcal{P}$. Then we may derive and add to our tableau the new goal obtained by replacing all occurrences of $\mathcal{P}$ in $\mathcal{F}$ with *true*, replacing all occurrences of $\mathcal{P}$ in $\mathcal{G}$ with *false*, and forming the conjunction of the results. The output entry associated with the derived goal is the conditional term whose test is the common subsentence $\mathcal{P}$ and whose *then*-clause and *else*-clause are the output entries s and t for $\mathcal{F}$ and $\mathcal{G}$, respectively. Because the resolution rule always introduces occurrences of the truth symbols *true* and *false* as proper subsentences, we can immediately apply *true-false* transformation rules to the derived goal.

Example

Suppose our tableau contains the rows

assertions	goals	outputs $sqrt(r, \epsilon)$
	$\boxed{max(r, 1) < \epsilon}^{+}$	0
	$not\ \boxed{max(r, 1) < \epsilon}^{-}$	*if* $[sqrt(r, 2\epsilon) + \epsilon]^2 \leq r$ *then* $sqrt(r, 2\epsilon) + \epsilon$ *else* $sqrt(r, 2\epsilon)$

These goals have a common subsentence $max(r, 1) < \epsilon$, indicated by boxes. Therefore we may derive and add to our tableau the new goal

	true and not false	*if* $max(r, 1) < \epsilon$ *then* 0 *else if* $[sqrt(r, 2\epsilon) + \epsilon]^2 \leq r$ *then* $sqrt(r, 2\epsilon) + \epsilon$ *else* $sqrt(r, 2\epsilon)$

By application of transformation rules, this goal reduces to

	true	*if* $max(r, 1) < \epsilon$ *then* 0 *else if* $[sqrt(r, 2\epsilon) + \epsilon]^2 \leq r$ *then* $sqrt(r, 2\epsilon) + \epsilon$ *else* $sqrt(r, 2\epsilon)$

Note that, because we have derived the goal *true* with a primitive output entry, this could be the final step in a square-root derivation. (In fact, however, this will not be the final step in our derivation of a square-root program.) ⌟

If one of the given goals has no output entry, the derived output entry is not a conditional term; it is simply the output entry of the other given goal. If neither given goal has an output entry, the derived goal has no output entry either. We do not require that the two given goals be distinct; we may apply the rule to a goal and itself.

We have presented the resolution rule as it applies to two goals. According to the duality property of tableaux, however, we may transform an assertion into a goal simply by negating it. Therefore, we can apply the rule to an assertion and a goal, or to two assertions.

The resolution rule may be restricted by a *polarity strategy*, according to which we need not apply the rule unless some occurrence of $\mathcal{P}$ in $\mathcal{F}$ is "positive" and some occurrence of $\mathcal{P}$ in $\mathcal{G}$ is "negative". (Here a subsentence of a tableau is regarded as positive or negative if it is within the scope of an even or odd number, respectively, of negation connectives *not*. Each assertion is considered to be within the scope of an implicit negation; thus, while goals are positive, assertions are negative. The *if*-clause $\mathcal{P}$ of a subsentence (*if* $\mathcal{P}$ *then* $\mathcal{Q}$) is considered to be within the scope of an additional implicit negation.) This strategy allows us to disregard many useless applications of the rule. The application in the previous example is in accordance with the polarity strategy; the boxed subsentence is positive in the first goal and negative in the second, as indicated by the annotation.

Let us show that the resolution rule is sound: that is, in a given model of the theory and for a given specification, the meaning of the tableau is the same before and after application of the rule. It actually suffices to show that, if the derived goal is true, then

at least one of the given goals is true and, if the given output entries are suitable, so is the derived output entry.

Suppose the derived goal ($\mathcal{F}[true]$ *and* $\mathcal{G}[false]$) is true. Then both its conjuncts $\mathcal{F}[true]$ and $\mathcal{G}[false]$ are true. We distinguish between two cases, depending on whether or not the common subsentence $\mathcal{P}$ is true or false. In the case in which $\mathcal{P}$ is true, the (ground) goal $\mathcal{F}[\mathcal{P}]$ has the same truth-value as the conjunct $\mathcal{F}[true]$; that is, $\mathcal{F}[\mathcal{P}]$ is true. In the case in which $\mathcal{P}$ is false, the goal $\mathcal{G}[\mathcal{P}]$ has the same truth-value as the conjunct $\mathcal{G}[false]$; that is, $\mathcal{G}[\mathcal{P}]$ is true. In either case, one of the two given goals, $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$, is true.

Now assume that the given output entries are suitable. To show that the derived output entry is suitable, we suppose that the derived goal is true and establish that the derived output entry satisfies the input-output condition. We have seen that, in the case in which $\mathcal{P}$ is true, the given goal $\mathcal{F}[\mathcal{P}]$ is true; because its output entry s is suitable, it satisfies the input-output condition. Similarly, in the case in which $\mathcal{P}$ is false, the output entry t satisfies the input-output condition. In either case, therefore, the conditional term (*if* $\mathcal{P}$ *then* s *else* t) satisfies the input-output condition; but this is the derived output entry.

THE RESOLUTION RULE: GENERAL VERSION

We have described the ground version of the resolution rule, which applies to goals with no variables. We now present the *general version*, which applies to goals with variables. In this case, we can apply a substitution to the goals, as necessary, to create a common subsentence.

assertions	goals	outputs $f(a)$
	$\mathcal{F}[\mathcal{P}]$	s
	$\mathcal{G}[\hat{\mathcal{P}}]$	t
	$\mathcal{F}\theta[true]$ *and* $\mathcal{G}\theta[false]$	*if* $\mathcal{P}\theta$ *then* $s\theta$ *else* $t\theta$

More precisely, suppose our tableau contains goals $\mathcal{F}$ and $\mathcal{G}$, which have no variables in common. (This can be ensured by renaming the variables of the rows as necessary, according to the *renaming* property.) Suppose further that some of the subsentences of $\mathcal{F}$ and some of the subsentences of $\mathcal{G}$ are unifiable, with a most-general unifier θ; let $\mathcal{P}\theta = \hat{\mathcal{P}}\theta$ be the unified subsentence. Then we may derive and add to our tableau the new goal

obtained by replacing all occurrences of $\mathcal{P}\theta$ in $\mathcal{F}\theta$ with *true*, replacing all occurrences of $\mathcal{P}\theta$ in $\mathcal{G}\theta$ with *false*, and forming the conjunction of the results. The associated output entry is a conditional term whose test is the unified subsentence $\mathcal{P}\theta$, and whose *then*-clause and *else*-clause are the corresponding instances $s\theta$ and $t\theta$, respectively, of the given output entries.

In other words, to apply the general version of the rule to $\mathcal{F}$ and $\mathcal{G}$, we apply the ground version of the rule to $\mathcal{F}\theta$ and $\mathcal{G}\theta$. The soundness of the general version can be deduced from the soundness of the ground version and the instance property. The polarity strategy applies as before. If we wish to apply the rule to an assertion and a goal or to two assertions, we can regard the assertions as goals by negating them, as in the ground case.

Example

Suppose our tableau contains the rows

assertions	goals	outputs $sqrt(r, \epsilon)$
	$\boxed{\hat{z}^2 \le r \text{ and } \text{not}\,[(\hat{z}+2\epsilon)^2 \le r]}^{+}$	*if* $(\hat{z}+\epsilon)^2 \le r$ *then* $\hat{z}+\epsilon$ *else* $\hat{z}$
if $\langle x, v\rangle \prec_w \langle r, \epsilon\rangle$ *then if* $0 \le x$ *and* $0 < v$ *then* $\boxed{(sqrt(x, v))^2 \le x \text{ and } \text{not}\,[(sqrt(x, v)+v)^2 \le x]}^{-}$		

The boxed subsentences are unifiable; a most-general unifier is

$$\theta: \quad \{x \leftarrow r,\ v \leftarrow 2\epsilon,\ \hat{z} \leftarrow sqrt(r, 2\epsilon)\}.$$

The subsentences have respectively positive and negative polarity, as indicated by the annotation. We may regard the assertion as a goal by negating it. By application of the general version of the resolution rule, we may derive the new row

	true *and* *not* [*if* $\langle r, 2\epsilon\rangle \prec_w \langle r, \epsilon\rangle$ *then if* $0 \le r$ *and* $0 < 2\epsilon$ *then false*]	*if* $[sqrt(r, 2\epsilon)+\epsilon]^2 \le r$ *then* $sqrt(r, 2\epsilon)+\epsilon$ *else* $sqrt(r, 2\epsilon)$

By the application of transformation rules, this goal reduces to

	$\langle r, 2\epsilon\rangle \prec_w \langle r, \epsilon\rangle$ *and* $0 \le r$ *and* $0 < 2\epsilon$	*if* $[sqrt(r, 2\epsilon) + \epsilon]^2 \le r$ *then* $sqrt(r, 2\epsilon) + \epsilon$ *else* $sqrt(r, 2\epsilon)$

Note that the unifier θ has been applied to all variables in the given rows, including those in the output entry. Because the given assertion has no output entry, no new conditional term is formed in deriving the output entry. This application of the rule is in accordance with the polarity strategy. ∎

Our resolution rule differs from the familiar resolution rule of Robinson [65] in that it is *nonclausal*; it applies to quantifier-free sentences with a full set of logical connectives that need not be in clausal form or any other normal form. Nonclausal resolution reduces to classical resolution in the clausal case. The nonclausal rule was developed independently by Manna and Waldinger [80] and Murray [82]. The resolution rule and the *true-false* transformation rules have been shown by Murray to constitute a complete system for first-order logic. The polarity strategy maintains this completeness.

We use an associative-commutative unification algorithm (as in Stickel [81]) so that the associative and commutative properties of such operators as addition and conjunction can be taken into account in finding a unifier; thus, $p\big(f(x) + (b + g(a))\big)$ can be unified with $p\big((g(y) + f(b)) + x\big)$.

The resolution rule accounts for the introduction of the notion of binary search into our derivation.

THE DISCOVERY OF BINARY SEARCH

Recall that our initial goal is

	2. $\boxed{z^2 \le r}^+$ *and not* $[(z+\epsilon)^2 \le r]$	z

We are about to apply the resolution rule to this goal and itself. To make this step easier to understand, let us write another copy of the goal.

	$\hat{2}$. $\hat{z}^2 \le r$ *and not* $\boxed{(\hat{z}+\epsilon)^2 \le r}^-$	$\hat{z}$

We have renamed the variable of the second copy of the goal so that, as required, the two copies have no variables in common.

The boxed subsentences of the two copies of the goal are unifiable; a most-general unifier is

$$\theta: \quad \{z \leftarrow \hat{z} + \epsilon\}.$$

Therefore, we can apply the resolution rule between the two copies of the goal to obtain

	true and not $[((\hat{z} + \epsilon) + \epsilon)^2 \leq r]$ *and* $\hat{z}^2 \leq r$ *and not false*	*if* $(\hat{z} + \epsilon)^2 \leq r$ *then* $\hat{z} + \epsilon$ *else* $\hat{z}$

By application of transformation rules, including the rule

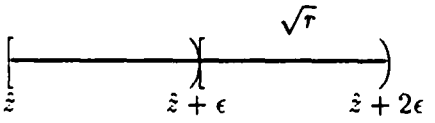
$$u + u \rightarrow 2u,$$

this goal can be reduced to

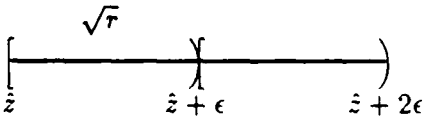
	3. $\hat{z}^2 \leq r$ *and* *not* $[(\hat{z} + 2\epsilon)^2 \leq r]$	*if* $(\hat{z} + \epsilon)^2 \leq r$ *then* $\hat{z} + \epsilon$ *else* $\hat{z}$

(We have reordered the conjuncts for pedagogical reasons only; because we use associative-commutative unification, their actual order is irrelevant.)

According to this goal, it suffices to find a rougher estimate $\hat{z}$, which is within a tolerance 2ϵ less than $\sqrt{r}$. For then either $\hat{z} + \epsilon$ or $\hat{z}$ itself will be within ϵ less than $\sqrt{r}$, depending on whether or not $\hat{z} + \epsilon$ is less than or equal to $\sqrt{r}$, that is, $(\hat{z} + \epsilon)^2 \leq r$. The two possibilities are illustrated below:



Case: $\hat{z} + \epsilon \leq \sqrt{r}$



Case: *not* $[\hat{z} + \epsilon \leq \sqrt{r}\,]$

Goal 3 contains the essential idea of binary search as applied to the square-root problem. Although the idea seems subtle to us, it appears almost immediately in the derivation. The step is nearly inevitable: any brute-force search procedure would discover it.

The derivation of the new goal is logically straightforward, but the intuition behind it may be a bit mysterious. Let us paraphrase the reasoning in a more geometric way. Our initial goal expresses the fact that it suffices to find a real number z such that $\sqrt{r}$ belongs to the half-open interval $[z, z + \epsilon)$. Our rewritten copy of this goal expresses the fact that it is equally acceptable to find a real number $\hat{z}$ such that $\sqrt{r}$ belongs to the half-open interval $[\hat{z}, \hat{z} + \epsilon)$. We shall be content to achieve either of these goals; i.e., we shall be

happy if $\sqrt{r}$ belongs to either of the two half-open intervals. In taking z to be $\hat{z} + \epsilon$, we are concatenating the two intervals, obtaining a new half-open interval $[\hat{z}, \hat{z} + 2\epsilon)$ twice the length of the original. It suffices to find a real number $\hat{z}$ such that $\sqrt{r}$ belongs to this new, longer interval, because then $\sqrt{r}$ must belong to one or the other of the two shorter ones.

THE THEORY RESOLUTION RULE

It is difficult to prevent a system from deriving numerous irrelevant consequences from the rows in a tableau. We can apply the resolution rule to virtually every goal in our derivation if our tableau contains an assertion such as ($u < v$ *or* $v \leq u$). Stickel [85] has introduced an extension to the resolution rule, which enables it to behave as if certain properties of the theory were "built in." This *theory resolution rule* does not add to the logical power of the system, but it does give us a heuristic advantage over a system in which all properties must be represented as assertions. When a property is built into the theory resolution rule, it is brought to bear only when it is appropriate.

The instance of Stickel's rule that we shall need is as follows. (Stickel's actual rule is more general.) Let us suppose that $\mathcal{H}[\mathcal{P}, \mathcal{Q}]$ is a valid sentence we wish to build in. Then the ground version of the *theory resolution rule*, *invoking the property* $\mathcal{H}[\mathcal{P}, \mathcal{Q}]$, is as follows:

assertions	goals	outputs $f(a)$
	$\mathcal{F}[\mathcal{P}]$	s
	$\mathcal{G}[\mathcal{Q}]$	t
	$\mathcal{F}[true]$ *and* $\mathcal{G}[true]$ *and* *not* $\mathcal{H}[false, false]$	*if* $\mathcal{P}$ *then* s *else* t

For strategic purposes, we may assume that $\mathcal{P}$ and $\mathcal{Q}$ are of positive polarity in the tableau and in $\mathcal{H}$. (In other words, they are within the scope of an even number of explicit or implicit negations in $\mathcal{H}$.) There are other versions of the rule that are strategically preferable if $\mathcal{P}$ or $\mathcal{Q}$ is negative. The soundness of the rule actually does not depend on the polarity.

The rule can be justified by adding the property $\mathcal{H}[\mathcal{P}, \mathcal{Q}]$ to the tableau as an assertion

$\mathcal{H}[\mathcal{P}, \boxed{\mathcal{Q}}^{-}]$		

(Note that because $\mathcal{Q}$ is positive in the assertion $\mathcal{H}$ and because each assertion is within the scope of an implicit negation, $\mathcal{Q}$ is negative in the tableau.) Applying the ordinary resolution rule to the goal

	$\mathcal{G}[\boxed{\mathcal{Q}}^+]$	t

and to this assertion, we obtain the new goal

	$\mathcal{G}[true]$ *and* *not* $\mathcal{H}[\boxed{\mathcal{P}}^-, false]$	t

Applying the resolution rule again, to the goal

	$\mathcal{F}[\boxed{\mathcal{P}}^+]$	s

and to the new goal, we obtain

	$\mathcal{F}[true]$ *and* $\mathcal{G}[true]$ *and* *not* $\mathcal{H}[false, false]$	*if* $\mathcal{P}$ *then* s *else* t

But this is precisely the conclusion drawn by the theory resolution rule, invoking the property $\mathcal{H}[\mathcal{P}, \mathcal{Q}]$.

We have just presented the ground version of the rule. To apply the general version, we first assume that the rows and the property $\mathcal{H}$ have no variables in common. We then apply a most-general unifier θ that allows the ground version of the rule to become applicable to $\mathcal{F}\theta$ and $\mathcal{G}\theta$, invoking $\mathcal{H}\theta$.

Example

Suppose our tableau contains the two goals

assertions	goals	outputs $sqrt(r, \epsilon)$
	$\boxed{max(r, 1) < \epsilon}^+$	0
	$\boxed{\epsilon \leq y}^+$	*if* $\left[sqrt(r, 2\epsilon) + \epsilon\right]^2 \leq r$ *then* $sqrt(r, 2\epsilon) + \epsilon$ *else* $sqrt(r, 2\epsilon)$

Suppose we have built into the theory resolution rule the sentence

$$\mathcal{H}: \boxed{u < v}^+ \text{ or } \boxed{v \leq u}^+.$$

The boxed subsentences of the two goals are unifiable with the correspondingly boxed subsentences of the sentence $\mathcal{H}$; a most-general unifier is

$$\theta: \quad \{u \leftarrow max(r, 1),\ v \leftarrow \epsilon,\ y \leftarrow max(r, 1)\}.$$

According to the theory resolution rule, we can obtain the new goal

	true and *true and* *not (false or false)*	*if* $max(r, 1) < \epsilon$ *then* 0 *else if* $[sqrt(r, 2\epsilon) + \epsilon]^2 \leq r$ *then* $sqrt(r, 2\epsilon) + \epsilon$ *else* $sqrt(r, 2\epsilon)$

which is transformed into

	true	*if* $max(r, 1) < \epsilon$ *then* 0 *else if* $[sqrt(r, 2\epsilon) + \epsilon]^2 \leq r$ *then* $sqrt(r, 2\epsilon) + \epsilon$ *else* $sqrt(r, 2\epsilon)$

(Note that this could be the final step in a square-root derivation.) ∎

We have introduced two additional rules to give special treatment to equality, orderings, and other important relations (Manna and Waldinger [86]), but these rules play no part in the portion of the derivation to be discussed in detail.

We shall now need the induction rule; this we describe in the next section.

4. RECURSION FORMATION

The rules presented so far do not allow us to introduce any repetitive construct into the program being derived. The mathematical-induction rule accounts for the introduction of recursion into the derived program.

We employ a single well-founded induction rule, which applies to a variety of theories.

THE MATHEMATICAL INDUCTION RULE

A well-founded relation $\prec_w$ is one that admits of no infinite decreasing sequences, i.e., sequences $x_1, x_2, x_3, \ldots$, such that

$$x_1 \succ_w x_2 \text{ and } x_2 \succ_w x_3 \text{ and} \ldots .$$

For instance, the less-than relation $<$ is well-founded in the theory of nonnegative integers, but not in the theory of real numbers.

The *well-founded induction rule* is expressed as follows. Suppose our initial tableau is

assertions	goals	outputs $f(a)$
$\mathcal{P}[a]$		
	$\mathcal{R}[a, z]$	z

In other words, we are attempting to construct a program f that, for an arbitrary input a, yields an output z satisfying the input-output condition

if $\mathcal{P}[a]$
then $\mathcal{R}[a, z]$.

According to the well-founded induction rule, we may prove this while assuming, as our induction hypothesis, that the program f will yield an output $f(x)$ satisfying the same input-output condition

if $\mathcal{P}[x]$
then $\mathcal{R}[x, f(x)]$,

provided that its input x is less than our original input a with respect to some well-founded relation $\prec_w$, that is, $x \prec_w a$. In other words, we may add to our tableau the new assertion

if $x \prec_w a$ *then if* $\mathcal{P}[x]$ *then* $\mathcal{R}[x, f(x)]$		

where x is a new variable. The well-founded relation $\prec_w$ used in the induction rule is arbitrary and must be selected later in the proof.

Example

The initial tableau in the square-root derivation is

assertions	goals	outputs $sqrt(r, \epsilon)$
$0 \leq r$ *and* $0 < \epsilon$		
	$z^2 \leq r$ *and* *not* $[(z+\epsilon)^2 \leq r]$	z

By application of the well-founded induction rule, we may introduce as a new assertion the induction hypothesis

if $\langle x, v\rangle \prec_w \langle r, \epsilon\rangle$ *then if* $0 \leq x$ *and* $0 < v$ *then* $(sqrt(x, v))^2 \leq x$ *and* *not* $[(sqrt(x, v)+v)^2 \leq x]$		

where x and v are variables. In other words, we may assume inductively that the output of the square-root program being constructed will satisfy the input-output condition for inputs x and v that are less than the given inputs r and ϵ with respect to some well-founded relation $\prec_w$. Because the program has two input parameters rather than one, the induction hypothesis refers to pairs of nonnegative integers rather than individual integers.

As it turns out, this particular induction hypothesis is never used in our square-root derivation. ∎

Use of the induction hypothesis in the proof may account for the introduction of a recursive call into the derived program. For instance, suppose that in our derivation we manage to develop a goal of the form

	$\mathcal{G}[\,\boxed{\mathcal{R}[s, \hat{z}]}^{+}\,]$	$t[\hat{z}]$

The boxed subsentences of this goal and the induction hypothesis,

if $x \prec_w a$ *then if* $\mathcal{P}[x]$ *then* $\boxed{\mathcal{R}[x, f(x)]}^{-}$		

are unifiable; a most-general unifier is

$$\theta: \quad \{x \leftarrow s,\ \hat{z} \leftarrow f(s)\}.$$

Therefore, we can apply the resolution rule to obtain the new goal

	$\mathcal{G}[true]$ *and* *not* $\left[\begin{array}{l} \textit{if } s \prec_w a \\ \textit{then if } \mathcal{P}[s] \\ \qquad \textit{then } \textit{false} \end{array}\right]$	$t[f(s)]$

This goal reduces under transformation to

	$\mathcal{G}[true]$ *and* $\mathcal{P}[s]$ *and* $s \prec_w a$	$t[f(s)]$

Note that a recursive call $f(s)$ has been introduced into the output entry as a result of this step. The condition $\mathcal{P}[s]$ in the goal ensures the legality of the argument s, i.e., that it satisfies the input condition of the desired program. The condition $s \prec_w a$ ensures that the evaluation of the recursive call cannot lead to a nonterminating computation. (If there were an infinite computation, we could construct a corresponding infinite sequence of arguments decreasing with respect to $\prec_w$, thus contradicting the definition of a well-founded relation.)

Example

In our square-root derivation we have developed the goal

	$\boxed{\begin{array}{l} \hat{z}^2 \le r \\ \quad \textit{and} \\ \textit{not } [(\hat{z}+2\epsilon)^2 \le r] \end{array}}^{+}$	$\begin{array}{l} \textit{if } (\hat{z}+\epsilon)^2 \le r \\ \textit{then } \hat{z}+\epsilon \\ \textit{else } \hat{z} \end{array}$

and the induction hypothesis

$\begin{array}{l} \textit{if } \langle x, v\rangle \prec_w \langle r, \epsilon\rangle \\ \textit{then if } 0 \le x \textit{ and } 0 < v \\ \qquad \textit{then } \boxed{\begin{array}{l} (sqrt(x, v))^2 \le x \textit{ and} \\ \textit{not } [(sqrt(x, v)+v)^2 \le x] \end{array}}^{-} \end{array}$			

The boxed subsentences are unifiable; a most-general unifier is

$$\theta: \quad \{x \leftarrow r,\ v \leftarrow 2\epsilon,\ \hat{z} \leftarrow sqrt(r, 2\epsilon)\}.$$

We obtain (after transformation)

	$\langle r, 2\epsilon\rangle \prec_w \langle r, \epsilon\rangle$ *and* $0 \leq r$ *and* $0 < 2\epsilon$	*if* $\left[sqrt(r, 2\epsilon) + \epsilon\right]^2 \leq r$ *then* $sqrt(r, 2\epsilon) + \epsilon$ *else* $sqrt(r, 2\epsilon)$

Note that at this point three recursive calls $sqrt(r, 2\epsilon)$ have been introduced into the output entry. The condition ($0 \leq r$ *and* $0 < 2\epsilon$) ensures that the arguments r and 2ϵ of these recursive calls will satisfy the input condition for the program, i.e, that r is nonnegative and 2ϵ is positive. The condition $\langle r, 2\epsilon\rangle \prec_w \langle r, \epsilon\rangle$ ensures that the newly introduced recursive calls cannot lead to a nonterminating computation. The well-founded relation $\prec_w$ that serves as the basis for the induction is as yet unspecified.

For reasons that will become clear, this step will not actually be part of our square-root derivation. ⌟

The particular well-founded relation $\prec_w$ referred to in the induction hypothesis is not yet specified; it is selected at a later stage of the proof. If we allow well-founded relations to be objects in our domain, we may regard the sentence $x \prec_w y$ as an abbreviation for $\prec(w, x, y)$; thus, w is a variable that may be replaced by a particular relation. We assume that the properties of many known well-founded relations (such as $\prec_{tree}$, the proper-subtree relation over trees) and of operations for combining them are among the assertions of our initial tableau.

The well-founded induction principle (from which the rule is derived) is universally quantified over all well-founded relations: it is surrounded by a quantifier $(\forall w)$. When the quantifiers are removed by skolemization, the input a of the program being constructed becomes a skolem term $a(w)$ rather than a constant a. (Those unfamiliar with skolemization are asked to accept this on faith.) This has the effect that the well-founded relation w cannot be chosen to depend on the input parameter $a(w)$ itself. In particular, w is not unifiable with any term containing an occurrence of $a(w)$. Otherwise the induction rule would be unsound and the termination argument sketched above would not apply. If we could alter the well-founded relation with each recursive call, we might indeed have an infinite computation. For simplicity of notation, however, we shall continue to write our input parameters as constants.

5. INTRODUCTION OF AUXILIARY SUBPROGRAMS

The induction rule, as we have presented it, can be applied only to the initial rows of a tableau. By the introduction of auxiliary subprograms, however, any rows of a tableau can be taken as the initial rows of a new tableau, to which we may apply the induction rule.

Suppose that in the course of a derivation we have obtained the rows

assertions	goals	outputs $f(a)$
$\tilde{\mathcal{P}}[s]$		
	$\boxed{\tilde{\mathcal{R}}[s, \tilde{z}]}^{+}$	$r[\tilde{z}]$

where s is a ground term and $\tilde{z}$ is a variable. Then we may consider introducing a new auxiliary subprogram $\tilde{f}(\tilde{a})$, whose specification is

$$\tilde{f}(\tilde{a}) \Leftarrow \text{find } \tilde{z} \text{ such that } \tilde{\mathcal{R}}[\tilde{a}, \tilde{z}],$$
$$\text{where } \tilde{\mathcal{P}}[\tilde{a}].$$

(If $\tilde{\mathcal{R}}$ contains several variables $\tilde{z}_1, \tilde{z}_2, \ldots, \tilde{z}_n$, we must construct several auxiliaries $\tilde{f}_1$, $\tilde{f}_2, \ldots, \tilde{f}_n$.)

Assuming that we shall succeed in constructing such an auxiliary, we add to our original tableau an assertion that the new subprogram always meets its specification; namely,

$\mathcal{Q}$: *if* $\tilde{\mathcal{P}}[x]$ *then* $\boxed{\tilde{\mathcal{R}}[x, \tilde{f}(x)]}^{-}$		

The auxiliary $\tilde{f}$ is taken to be primitive. By application of the resolution rule to the goal $\tilde{\mathcal{R}}[s, \tilde{z}]$ and the new assertion, we obtain (after *true-false* transformation)

	$\tilde{\mathcal{P}}[s]$	$r[\tilde{f}(s)]$

By resolution of this goal with the assertion $\tilde{\mathcal{P}}[s]$, we obtain (after *true-false* transformation)

	true	$r[\tilde{f}(s)]$

If $r[\tilde{f}(s)]$ is primitive, this may be taken to be the final step in a derivation of $f(a)$. The

program we obtain is simply

$$f(a) \Leftarrow r[\tilde{f}(s)].$$

In adding the new assertion Q, however, we are incurring the obligation to construct a suitable auxiliary subprogram $\tilde{f}(\tilde{a})$. For this purpose, we introduce a new tableau, whose initial rows are

assertions	goals	outputs $\tilde{f}(\tilde{a})$
$\tilde{\mathcal{P}}[\tilde{a}]$		
	$\tilde{\mathcal{R}}[\tilde{a}, \tilde{z}]$	$\tilde{z}$

Because this is an initial tableau, we may apply the induction rule to add the induction hypothesis

if $u \prec_w \tilde{a}$ *then if* $\tilde{\mathcal{P}}[u]$ *then* $\tilde{\mathcal{R}}[u, \tilde{f}(u)]$		

We can actually form auxiliary subprograms whose input condition is a conjunction ($\tilde{\mathcal{P}}_1$ *and* $\tilde{\mathcal{P}}_2$ *and* ...) of assertions and whose output condition is a disjunction ($\tilde{\mathcal{R}}_1$ *or* $\tilde{\mathcal{R}}_2$ *or* ...) of goals, but we can do without this complication here.

We shall defer giving an example of auxiliary-subprogram introduction until we have discussed the strategic controls for such a step.

STRATEGIC CONSIDERATIONS

Adding a new auxiliary subprogram is not without risk, because it can happen that there is no program meeting the specification of the auxiliary even though the original programming problem does have a solution. Although we are not primarily concerned with the heuristic aspects of program synthesis in this paper, we shall mention the heuristic indicators for introducing the auxiliary.

In the course of the main derivation, suppose we have obtained the rows

assertions	goals	outputs $f(a)$
$\tilde{\mathcal{P}}[s]$		
	$\tilde{\mathcal{R}}[s, \tilde{z}]$	$r[\tilde{z}]$

as before. What indicates that we should take these rows as the specification for a new subprogram?

Assume that, only from these rows and assertions representing valid sentences of the theory, we obtain a goal of the form

	$\mathcal{G}[\tilde{\mathcal{R}}[t, \hat{z}]]$	$q[\hat{z}]$

where t is a term, $\hat{z}$ a variable, and $\tilde{\mathcal{R}}$ is positive in $\mathcal{G}$. In other words, the new goal contains as a subsentence a "replica" $\tilde{\mathcal{R}}[t, \hat{z}]$ of the higher-level goal $\tilde{\mathcal{R}}[s, \tilde{z}]$. The replica is obtained by replacing a term s of the goal with a different term t and the variable $\tilde{z}$ with a possibly different variable $\hat{z}$.

This suggests forming an auxiliary $\tilde{f}(\tilde{a})$ with input condition $\tilde{\mathcal{P}}[\tilde{a}]$ and output condition $\tilde{\mathcal{R}}[\tilde{a}, \tilde{z}]$, where $\tilde{a}$ is a new constant, the input parameter of the subprogram. The initial tableau for the auxiliary is

assertions	goals	outputs $\tilde{f}(\tilde{a})$
$\tilde{\mathcal{P}}[\tilde{a}]$		
	$\tilde{\mathcal{R}}[\tilde{a}, \tilde{z}]$	$\tilde{z}$

If we succeed in imitating the original derivation in the auxiliary tableau and developing a corresponding subgoal of the form

	$\mathcal{G}[\,\boxed{\tilde{\mathcal{R}}[\tilde{t}, \hat{z}]}^{+}\,]$	$\tilde{q}[\hat{z}]$

we can then apply the resolution rule to this goal and the induction hypothesis for the auxiliary.

if $u \prec_w \tilde{a}$ *then if* $\tilde{\mathcal{P}}[u]$ *then* $\boxed{\tilde{\mathcal{R}}[u, \tilde{f}(u)]}^{-}$		

The unifying substitution is

$$\{u \leftarrow \tilde{t},\ \hat{z} \leftarrow \tilde{f}(\tilde{t})\}.$$

We obtain (after transformation)

	$\mathcal{G}[true]$ *and* $\tilde{t} \prec_w \tilde{a}$ *and* $\tilde{\mathcal{P}}[\tilde{t}]$	$\tilde{q}[\tilde{f}(\tilde{t})]$

In other words, the appearance of a replica in the main derivation suggests that we form the appropriate auxiliary, so that in the auxiliary derivation we shall be able to unify the corresponding replica with the conclusion of the auxiliary induction hypothesis. There is, of course, the unfortunate possibility that we shall not be able to obtain the appropriate replica in the auxiliary derivation, because we have replaced a term s in the main derivation with the new constant $\tilde{a}$ in the auxiliary. If the original derivation relies on special properties of s, we may not be able to imitate it with the constant $\tilde{a}$.

During the derivation of the auxiliary, we may discover that we require a new assertion $\tilde{\mathcal{P}}'[\tilde{a}]$, where $\tilde{\mathcal{P}}'[s]$ is already an assertion in our original tableau. In this case, we may attempt to add $\tilde{\mathcal{P}}'[\tilde{a}]$ as an input condition to the auxiliary specification, to obtain

$$\tilde{f}(\tilde{a}) \Leftarrow \text{find } \tilde{z} \text{ such that } \tilde{\mathcal{R}}[\tilde{a}, \tilde{z}]$$
$$\text{where } \tilde{\mathcal{P}}[\tilde{a}] \text{ and } \tilde{\mathcal{P}}'[\tilde{a}].$$

We may then add the new condition to the initial assertion in the auxiliary tableau, to obtain

$\tilde{\mathcal{P}}[\tilde{a}]$ *and* $\tilde{\mathcal{P}}'[\tilde{a}]$		

We must make corresponding alterations in the induction hypothesis for the auxiliary tableau, in those portions of the proof that use the induction hypothesis, and in the assertion describing the auxiliary in the main tableau. Thus the precise specification of the auxiliary may be built up incrementally, after the derivation of the subprogram is under way.

In Manna and Waldinger [80], we introduced auxiliary subprograms by adding a new output column in the original tableau rather than adding a new tableau. Traugott [86] uses multiple tableaux to introduce subprograms, as we do here.

SQUARE ROOT: INTRODUCTION OF THE SUBPROGRAM

In the tableau for the square-root derivation, we are initially given the rows

assertions	goals	outputs $sqrt(r, \epsilon)$
1. $0 \leq r$ *and* $0 < \epsilon$		
	2. $\boxed{\begin{array}{l} z^2 \leq r \text{ and} \\ \text{not } [(z+\epsilon)^2 \leq r] \end{array}}^+$	z

By resolving the goal with itself and transforming, we have obtained the subgoal

	3. $\hat{z}^2 \leq r$ *and* *not* $[(\hat{z}+2\epsilon)^2 \leq r]$	*if* $(\hat{z}+\epsilon)^2 \leq r$ *then* $\hat{z}+\epsilon$ *else* $\hat{z}$

The entire subgoal is a replica of the initial goal, obtained by replacing the term ϵ with 2ϵ and the variable z with $\hat{z}$. This suggests introducing a new auxiliary subprogram $\widetilde{sqrt}(\tilde{\epsilon})$, whose parameter $\tilde{\epsilon}$ plays the role of the replaced term ϵ in the initial goal, and whose input and output conditions are the initial assertion and goal, with ϵ replaced by $\tilde{\epsilon}$; that is,

$$\widetilde{sqrt}(\tilde{\epsilon}) \Leftarrow \text{find } \tilde{z} \text{ such that } \tilde{z}^2 \leq r \text{ and not } [(\tilde{z}+\tilde{\epsilon})^2 \leq r]$$
$$\text{where } 0 \leq r \text{ and } 0 < \tilde{\epsilon}.$$

We do not include a parameter $\tilde{r}$ in the auxiliary because r was not replaced in forming the replica. For the auxiliary, r is global rather than a parameter. When $\widetilde{sqrt}$ is evaluated, r will be bound to an argument of the main program *sqrt*.

The initial assertion ($0 \leq r$ *and* $0 < \epsilon$) in the main tableau was not actually used in developing the replica. However, the corresponding initial assertion ($0 \leq r$ *and* $0 < \tilde{\epsilon}$) turns out to be necessary to complete the derivation of the auxiliary. In an automated implementation, this condition would most likely be added to the input condition for the auxiliary incrementally, after the derivation of the auxiliary was under way.

Assuming that we shall succeed in the synthesis of the auxiliary $\widetilde{sqrt}$, we add to our main tableau the assertion that $\widetilde{sqrt}$ does indeed meet its specification for all inputs v; that is,

4. *if* $0 \leq r$ *and* $0 < v$ *then* $\boxed{\begin{array}{l} (\widetilde{sqrt}(v))^2 \leq r \text{ and} \\ \text{not } [(\widetilde{sqrt}(v)+v)^2 \leq r] \end{array}}^-$		

By resolving the initial goal 2 with this assertion, and then resolving the resulting goal with the initial assertion, we obtain (after *true-false* transformation) the final goal

	5. *true*	$\widetilde{sqrt}(\epsilon)$

Note that the goal 3, which serves to suggest introducing the auxiliary $\widetilde{sqrt}$, turns out to play no part in the derivation of the main program. The main program we obtain is simply

$$sqrt(r, \epsilon) \Leftarrow \widetilde{sqrt}(\epsilon).$$

The only difference between the main program $sqrt(r, \epsilon)$ and the auxiliary $\widetilde{sqrt}(\tilde{\epsilon})$ is that r is a parameter for *sqrt* but not for $\widetilde{sqrt}$. This turns out to be a crucial distinction, however, because the well-founded relation we employ in the derivation of $\widetilde{sqrt}$ depends on r. The well-founded relation for a program cannot depend on a parameter for that program; otherwise the induction is not sound and termination is jeopardized. Had we not introduced the auxiliary, we would not have been able to complete this derivation. (Other derivations would be possible, using more artificial well-founded relations.)

6. COMPLETION OF THE SQUARE-ROOT DERIVATION

In this section we apply the principles we have introduced to complete the derivation of the square-root subprogram.

INTRODUCTION OF THE RECURSIVE CALL

In deriving the auxiliary, we begin with the tableau

assertions	goals	outputs $\widetilde{sqrt}(\tilde{\epsilon})$
$\tilde{1}$. $0 \leq r$ *and* $0 < \tilde{\epsilon}$		
	$\tilde{2}$. $\tilde{z}^2 \leq r$ *and* *not* $[(\tilde{z} + \tilde{\epsilon})^2 \leq r]$	$\tilde{z}$

We attempt to mimic the main derivation. Resolving the initial goal with itself and transforming as before, we obtain

	$\tilde{3}$. $\boxed{\hat{z}^2 \leq r \text{ and } \textit{not } [(\hat{z} + 2\tilde{\epsilon})^2 \leq r]}^+$	*if* $(\hat{z} + \tilde{\epsilon})^2 \leq r$ *then* $\hat{z} + \tilde{\epsilon}$ *else* $\hat{z}$

We assume inductively that the auxiliary $\widetilde{sqrt}$ will satisfy its specification for all inputs v less than its parameter $\tilde{\epsilon}$, with respect to some well-founded relation $\prec_w$, i.e.,

$\tilde{4}$. *if* $v \prec_w \tilde{\epsilon}$ *then if* $0 \le r$ *and* $0 < v$ *then* $\boxed{\begin{array}{l}\left(\widetilde{sqrt}(v)\right)^2 \le r \text{ and} \\ \text{not } \left[\left(\widetilde{sqrt}(v)+v\right)^2 \le r\right]\end{array}}$		

The boxed subsentences of the goal $\tilde{3}$ and the induction hypothesis $\tilde{4}$ are unifiable; a most-general unifier is

$$\{v \leftarrow 2\tilde{\epsilon},\ \hat{z} \leftarrow \widetilde{sqrt}(2\tilde{\epsilon})\}.$$

Applying the resolution rule, we obtain (after transformation)

	$\tilde{5}$. $2\tilde{\epsilon} \prec_w \tilde{\epsilon}$ *and* $0 \le r$ *and* $0 < 2\tilde{\epsilon}$	*if* $\left[\widetilde{sqrt}(2\tilde{\epsilon})+\tilde{\epsilon}\right]^2 \le r$ *then* $\widetilde{sqrt}(2\tilde{\epsilon})+\tilde{\epsilon}$ *else* $\widetilde{sqrt}(2\tilde{\epsilon})$

This step accounts for the introduction of three instances of a recursive call $\widetilde{sqrt}(2\tilde{\epsilon})$ into the auxiliary subprogram. As before, the condition ($0 \le r$ *and* $0 < 2\tilde{\epsilon}$) ensures that the argument $2\tilde{\epsilon}$ of this recursive call will satisfy the input condition. The condition $2\tilde{\epsilon} \prec_w \tilde{\epsilon}$ ensures that the newly introduced recursive call cannot lead to a nonterminating computation. The well-founded relation $\prec_w$ is as yet unspecified.

THE CHOICE OF THE WELL-FOUNDED RELATION

We have assumed that the definitions and properties of well-founded relations over several domains, including the real numbers, are among the assertions of our tableau. The relation to be selected in this derivation is the *bounded-doubling* relation $\prec_{bd(y)}$, defined on the positive reals so that

$$u \prec_{bd(y)} v \quad \text{if and only if} \quad u = 2v \text{ and } v \le y,$$

for some fixed upper bound y. Thus, with respect to this relation, $2v$ is actually less than v. The upper bound y is a parameter of the relation: for each real number y, we obtain a different relation $\prec_{bd(y)}$.

The bounded-doubling relation is well-founded because we cannot double a positive real forever without exceeding the bound y; thus, with respect to this relation, no infinite

decreasing sequences exist. Note that we could have replaced the constant 2 with any real constant greater than 1 or, indeed, with a variable x, which would then become an additional parameter for bd; but we shall not require such generality here. Also, for $u \prec_{bd(y)} v$ to be true, we require that $v \leq y$ but not that $u \leq y$.

The property of the bounded-doubling relation we employ is

if $0 < v$ *and* $v \leq y$ *then* $\boxed{2v \prec_{bd(y)} v}^-$		

Recall that we regard $u \prec_w v$ as an abbreviation of $\prec (w, u, v)$. The boxed subsentences of our goal

	$\tilde{5}$. $\boxed{2\tilde{\epsilon} \prec_w \tilde{\epsilon}}^+$ *and* $0 \leq r$ *and* $0 < 2\tilde{\epsilon}$	*if* $\left[\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}\right]^2 \leq r$ *then* $\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}$ *else* $\widetilde{sqrt}(2\tilde{\epsilon})$

and the above assertion unify; a most-general unifier is

$$\{v \leftarrow \tilde{\epsilon},\ w \leftarrow bd(y)\}.$$

By resolution of the goal with the assertion, we obtain

	$\tilde{6}$. $0 \leq r$ *and* $0 < 2\tilde{\epsilon}$ *and* $0 < \tilde{\epsilon}$ *and* $\tilde{\epsilon} \leq y$	*if* $\left[\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}\right]^2 \leq r$ *then* $\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}$ *else* $\widetilde{sqrt}(2\tilde{\epsilon})$

At this stage, the well-founded relation $\prec_w$ has been chosen to be the bounded-doubling relation $\prec_{bd(y)}$. The upper bound y is as yet undetermined.

The rest of the derivation relies on the special-relation rules (Manna and Waldinger [86]), which we have not presented here, and is relatively straightforward. We shall not give it in detail, but we would like to give the intuitive argument, indicating some of the properties we use but not what rules we apply.

With the help of the initial assertion for the auxiliary,

$\tilde{1}$. $0 \leq r$ *and* $0 < \tilde{\epsilon}$		

we can discard the first three conjuncts of our goal $\tilde{6}$, leaving

	$\tilde{7}.\ \ \tilde{\epsilon} \leq y$	*if* $\left[\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}\right]^2 \leq r$ *then* $\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}$ *else* $\widetilde{sqrt}(2\tilde{\epsilon})$

We shall refer to this as the *upper-bound* goal. It maintains that, if we can find some upper bound y on our input parameter $\tilde{\epsilon}$, its output entry meets the specification.

Note that, because $\tilde{\epsilon}$ is a parameter, it was initially our abbreviation for a skolem term $\tilde{\epsilon}(w)$. Then the well-founded relation w was taken to be $bd(y)$, so $\tilde{\epsilon}$ in the upper-bound goal $\tilde{\epsilon} \leq y$ stands for $\tilde{\epsilon}(bd(y))$. Thus, this goal $\tilde{\epsilon}(bd(y)) \leq y$ is not unifiable with the reflexivity assertion $u \leq u$ – they have no common instance – and we are prevented from resolving them. In other words, we (fortunately) cannot take the upper bound on $\tilde{\epsilon}$ to be $\tilde{\epsilon}$ itself.

Let us set the *upper-bound* goal aside for the moment; its proof depends on our treatment of the base case, which we consider next.

THE BASE CASE

Recall that the initial goal for the auxiliary procedure $\widetilde{sqrt}(\tilde{\epsilon})$ is

	$\tilde{2}.\ \ \tilde{z}^2 \leq r$ *and not* $\left[(\tilde{z} + \tilde{\epsilon})^2 \leq r\right]$	$\tilde{z}$

We employ the initial condition $0 \leq r$ and properties of the reals (including $0 \cdot v = v$), taking $\tilde{z}$ to be 0, to reduce the goal to *not* $(\tilde{\epsilon}^2 \leq r)$, that is,

	$\tilde{8}.\ \ r < \tilde{\epsilon}^2$	0

Note that at this stage the output entry has become 0.

We next employ the transitivity of the less-than relation and the property

if $1 < u$ *then* $u < u^2$

to decompose our goal further, to $(r < \tilde{\epsilon}$ *and* $1 < \tilde{\epsilon})$, that is,

	$\tilde{9}.\ \ max(r, 1) < \tilde{\epsilon}$	0

In other words, in the case in which $max(r, 1) < \tilde{\epsilon}$, the output 0 will satisfy the input-output specification.

PROVING THE UPPER-BOUND GOAL

Because we have introduced the goal $max(r,1) < \tilde{\epsilon}$, we can restrict our attention to the case in which *not* $[max(r,1) < \tilde{\epsilon}]$, that is, $\tilde{\epsilon} \leq max(r,1)$. But in this case the upper bound y for our bounded-doubling relation $\prec_{bd(y)}$ can be taken to be $max(r,1)$ itself. Formally speaking, we apply the theory resolution rule to this goal $\tilde{9}$ and our *upper-bound* goal

	$\tilde{7}$. $\tilde{\epsilon} \leq y$	*if* $\left[\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}\right]^2 \leq r$ *then* $\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}$ *else* $\widetilde{sqrt}(2\tilde{\epsilon})$

We invoke the property ($u < v$ *or* $v \leq u$) and take the most-general unifier to be

$$\{u \leftarrow max(r,\ 1),\ v \leftarrow \tilde{\epsilon},\ y \leftarrow max(r,\ 1)\}.$$

We obtain the final goal

	$\widetilde{10}$. *true*	*if* $max(r,\ 1) < \tilde{\epsilon}$ *then* 0 *else if* $\left[\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}\right]^2 \leq r$ *then* $\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}$ *else* $\widetilde{sqrt}(2\tilde{\epsilon})$

The new conditional in the output entry is introduced by the theory resolution rule.

At this stage we can see why the introduction of an auxiliary in which r is not a parameter was necessary for this derivation. Had we retained r as a parameter, it would have appeared in the initial goal as a skolem function $\tilde{r}(w)$. Because w was subsequently replaced by $bd(y)$, the occurrence of r in the goal $max(r,1) < \tilde{\epsilon}$ would have become $\tilde{r}(bd(y))$. We would have been prevented from unifying y with the term $max(\tilde{r}(bd(y)),1)$, which contains y; this last step could therefore not have been performed. From an intuitive point of view, if r were not a parameter, the system would suspect that r might be increased with each recursive call. There might then be no upper bound for the bounded-doubling relation, and termination would not be guaranteed.

We have completed the derivation of the main program and the auxiliary. The final program we obtain is therefore

$$sqrt(r, \epsilon) \Leftarrow \widetilde{sqrt}(\epsilon)$$

$$\widetilde{sqrt}(\tilde{\epsilon}) \Leftarrow \begin{cases} \textit{if } max(r, 1) < \tilde{\epsilon} \\ \textit{then } 0 \\ \textit{else if } \left[\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}\right]^2 \leq r \\ \quad \textit{then } \widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon} \\ \quad \textit{else } \widetilde{sqrt}(2\tilde{\epsilon}) \end{cases}$$

7. SUMMARY

At this point we reproduce the entire square-root derivation, again omitting some straightforward steps.

MAIN PROGRAM

The initial tableau:

assertions	goals	outputs $sqrt(r, \epsilon)$
1. $0 \leq r$ *and* $0 < \epsilon$		
	2. $z^2 \leq r$ *and* *not* $[(z + \epsilon)^2 \leq r]$	z

By resolution applied to goal 2 and itself:

	3. $\hat{z}^2 \leq r$ *and* *not* $[(\hat{z} + 2\epsilon)^2 \leq r]$	*if* $(\hat{z} + \epsilon)^2 \leq r$ *then* $\hat{z} + \epsilon$ *else* $\hat{z}$

By auxiliary-procedure introduction:

4. *if* $0 \leq r$ *and* $0 < v$ *then* $\left(\widetilde{sqrt}(v)\right)^2 \leq r$ *and* *not* $\left[\left(\widetilde{sqrt}(v) + v\right)^2 \leq r\right]$		

This step has been motivated by the replication of goal 2 in goal 3.

By resolution, from goal 2, assertion 4, and assertion 1:

	5. *true*	$\widetilde{sqrt}(\epsilon)$

AUXILIARY SUBPROGRAM

The initial tableau:

assertions	goals	outputs $\widetilde{sqrt}(\bar{\epsilon})$
$\bar{1}$. $0 \leq r$ *and* $0 < \bar{\epsilon}$		
	$\bar{2}$. $\bar{z}^2 \leq r$ *and* *not* $[(\bar{z}+\bar{\epsilon})^2 \leq r]$	$\bar{z}$

By resolution applied to goal $\bar{2}$ and itself:

	$\bar{3}$. $\hat{z}^2 \leq r$ *and* *not*$[(\hat{z}+2\bar{\epsilon})^2 \leq r]$	*if* $(\hat{z}+\bar{\epsilon})^2 \leq r$ *then* $\hat{z}+\bar{\epsilon}$ *else* $\hat{z}$

The induction hypothesis:

$\bar{4}$. *if* $v \prec_\omega \bar{\epsilon}$ *then if* $0 \leq r$ *and* $0 < v$ *then* $\left(\widetilde{sqrt}(v)\right)^2 \leq r$ *and* *not* $\left[\left(\widetilde{sqrt}(v)+v\right)^2 \leq r\right]$		

By resolution applied to goal $\bar{3}$ and assertion $\bar{4}$:

	$\bar{5}$. $2\bar{\epsilon} \prec_\omega \bar{\epsilon}$ *and* $0 \leq r$ *and* $0 < 2\bar{\epsilon}$	*if* $\left[\widetilde{sqrt}(2\bar{\epsilon})+\bar{\epsilon}\right]^2 \leq r$ *then* $\widetilde{sqrt}(2\bar{\epsilon})+\bar{\epsilon}$ *else* $\widetilde{sqrt}(2\bar{\epsilon})$

Here the recursive calls have been introduced.

A property of the bounded-doubling relation:

if $0 < v$ *and* $v \leq y$ *then* $2v \prec_{bd(y)} v$		

By resolution applied to goal $\tilde{5}$ and the above property:

	$\tilde{6}$. $0 \leq r$ *and* $0 < 2\tilde{\epsilon}$ *and* $0 < \tilde{\epsilon}$ *and* $\tilde{\epsilon} \leq y$	*if* $\left[\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}\right]^2 \leq r$ *then* $\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}$ *else* $\widetilde{sqrt}(2\tilde{\epsilon})$

At this stage the well-founded relation is taken to be the bounded-doubling relation.

By resolution and special-relation rules, from goal $\tilde{6}$, assertion $\tilde{1}$, and properties of the reals:

	$\tilde{7}$. $\tilde{\epsilon} \leq y$	*if* $\left[\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}\right]^2 \leq r$ *then* $\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}$ *else* $\widetilde{sqrt}(2\tilde{\epsilon})$

By resolution and special-relation rules, from goal $\tilde{2}$, assertion $\tilde{1}$, and properties of the reals:

	$\tilde{8}$. $r < \tilde{\epsilon}^2$	0

Here the output $\tilde{z}$ has been taken to be 0.

A property of the reals:

if $1 < u$ *then* $u < u^2$		

By special-relation rules, from goal $\tilde{8}$, the above property, and others:

	$\tilde{9}$. $max(r, 1) < \tilde{\epsilon}$	0

By theory resolution, invoking the property ($u < v$ *or* $v \leq u$), applied to goals $\tilde{9}$ and $\tilde{7}$:

	$\widetilde{10}$. *true*	*if* $max(r, 1) < \tilde{\epsilon}$ *then* 0 *else if* $\left[\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}\right]^2 \leq r$ *then* $\widetilde{sqrt}(2\tilde{\epsilon}) + \tilde{\epsilon}$ *else* $\widetilde{sqrt}(2\tilde{\epsilon})$

At this stage, a suitable upper bound for the bounded-doubling relation has been found to be $max(r, 1)$.

The real-number square-root derivation was first discovered manually; it was subsequently reproduced with an interactive program-synthesis system.

8. VARIATIONS

In this section we present several analogous binary-search derivations for different problems and for different specifications of the same problem.

OTHER SQUARE-ROOT SPECIFICATIONS

It may have occurred to the reader that we were just lucky in our choice of specification, in that two subsentences of the output condition turned out to be unifiable. What if the specification had been in some other form? Would we have been able to obtain the same program?

For example, suppose we had phrased the output condition as

$$z^2 \leq r \ \textit{and} \ (z + \epsilon)^2 > r$$

or

$$z^2 \leq r \ \textit{and} \ r < (z + \epsilon)^2$$

instead of

$$z^2 \leq r \ \textit{and} \ \textit{not} \ [(z + \epsilon)^2 \leq r].$$

Then we would not have been able to unify the two subsentences of the initial goal and apply the resolution rule, as we did in our original derivation. How could we have proceeded?

In fact, we can apply the theory resolution rule, invoking the property

$$u \leq v \ \textit{or} \ u > v$$

or, respectively,

$$u \leq v \ \textit{or} \ v < u.$$

We obtain (after transformation) the new goal

$$\hat{z}^2 \leq r \ \textit{and} \ (\hat{z} + 2\epsilon)^2 > r$$

or, respectively,

$$\hat{z}^2 \leq r \ \textit{and} \ r < (\hat{z} + 2\epsilon)^2,$$

each of which is a replica of the initial goal. The balance of the derivations are as before.

We could also have phrased the output condition as

$$|\sqrt{r} - z| < \epsilon.$$

Here $\sqrt{r}$ is the precise square root of r; the function $\sqrt{u}$ is a nonprimitive that can nevertheless be employed in specification. This specification is weaker than the one we were given originally, since it permits z to be larger than $\sqrt{r}$. With the help of the input condition, properties of the absolute value function, and other properties of the reals, we can develop the goal

$$0 \leq \sqrt{r} - z \ \textit{and} \ \sqrt{r} - z < \epsilon$$

and then

$$0 \leq z \ \textit{and} \ z^2 \leq r \ \textit{and} \ r < (z + \epsilon)^2.$$

From this goal, we can derive the same program as before. Of course, because the specification is weaker, we can obtain a broader class of programs.

Many binary-search algorithms can be derived in an analogous way. Let us first consider some other real-number problems.

THE DIVISION ALGORITHM

Suppose a program to perform real-number division is specified as follows:

$$\begin{aligned} \textit{div}(r, s, \epsilon) \ \Leftarrow \ & \text{find } z \text{ such that} \\ & z \cdot s \leq r \ \textit{and not} \ [(z + \epsilon) \cdot s \leq r] \\ & \text{where } 0 \leq r \ \textit{and} \ 0 < s \ \textit{and} \ 0 < \epsilon. \end{aligned}$$

In other words, the program is required to yield a real number z that is within a tolerance ϵ less than r/s, the exact quotient of dividing r by s. We obtain the program

$$div(r, s, \epsilon) \Leftarrow \widetilde{div}(\epsilon)$$

$$\widetilde{div}(\tilde{\epsilon}) \Leftarrow \begin{cases} \textit{if } r < \tilde{\epsilon} \cdot s \\ \textit{then } 0 \\ \textit{else if } \left[\widetilde{div}(2\tilde{\epsilon}) + \tilde{\epsilon}\right] \cdot s \leq r \\ \quad \textit{then } \widetilde{div}(2\tilde{\epsilon}) + \tilde{\epsilon} \\ \quad \textit{else } \widetilde{div}(2\tilde{\epsilon}) \end{cases}$$

The auxiliary subprogram $\widetilde{div}$, which is analogous to the auxiliary subprogram $\widetilde{sqrt}$, is like the top-level division program *div* but takes r and s to be globals, not parameters. It meets the specification

$$\widetilde{div}(\tilde{\epsilon}) \Leftarrow \text{find } \tilde{z} \text{ such that } \tilde{z} \cdot s \leq r \textit{ and not } \left[(\tilde{z} + \tilde{\epsilon}) \cdot s \leq r\right]$$
$$\text{where } 0 \leq r \text{ and } 0 < s \text{ and } 0 < \tilde{\epsilon}.$$

The rationale for the $\widetilde{div}$ program, like its derivation, is analogous to that for the real-number square root. The program first checks whether the error tolerance is very big, that is, if $r < \tilde{\epsilon} \cdot s$. If so, the output can safely be taken to be 0. For, because $0 \leq r$, we have

$$0 \cdot s \leq r.$$

And, because $r < \tilde{\epsilon} \cdot s$, we have $r < (0 + \tilde{\epsilon}) \cdot s$, that is,

$$\textit{not } \left[(0 + \tilde{\epsilon}) \cdot s \leq r\right].$$

Thus, 0 satisfies both conjuncts of the output condition for $\widetilde{div}$ in this case.

On the other hand, if $\tilde{\epsilon}$ is small, that is, if $\tilde{\epsilon} \cdot s \leq r$, the program finds a rougher estimate $\widetilde{div}(2\tilde{\epsilon})$, which is within $2\tilde{\epsilon}$ less than r/s. The program considers whether increasing this estimate by $\tilde{\epsilon}$ will leave it less than r/s. If so, the rough estimate may be increased by $\tilde{\epsilon}$; if not, the rough estimate is already close enough.

The termination proof for this program is also analogous to that for the square root. Although the argument $\tilde{\epsilon}$ is doubled with each recursive call, the other arguments are unchanged and the calls are evaluated only in the case in which $\tilde{\epsilon} \cdot s \leq r$, that is, $\tilde{\epsilon} \leq r/s$. Thus, there is a uniform upper bound on the doubled argument.

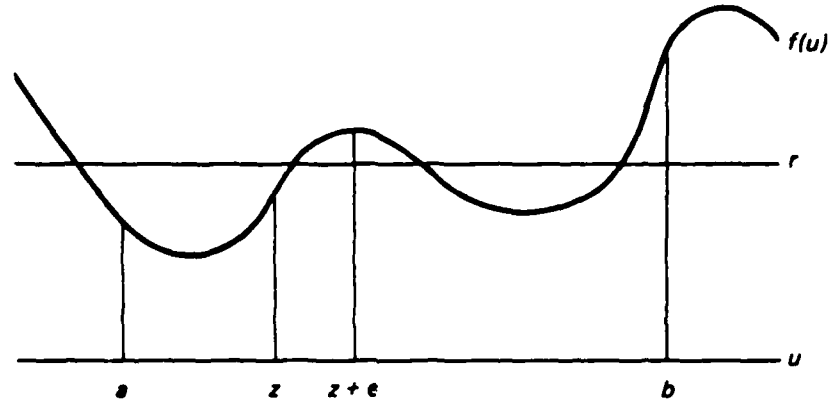
BINARY SEARCH SCHEMATA

It may be clear from the foregoing discussion that there is little in the derivations for the square-root and division programs that depends on the properties of these functions. More or less the same derivation suffices for finding an approximate solution to an arbitrary real-number equation $f(z) = r$.

For a given primitive function symbol f, we consider the specification

$$solve(r, \epsilon) \Leftarrow \text{find } z \text{ such that } f(z) \leq r \text{ and not } [f(z+\epsilon) \leq r]$$
$$\text{where } f(a) \leq r \text{ and } \begin{bmatrix} \text{if } b < u \\ \text{then not } (f(u) \leq r) \end{bmatrix} \text{ and } 0 < \epsilon.$$

Here a and b are primitive constants and u is a variable. In other words, we assume that there exist real numbers a and b such that $f(a) \leq r$ and $f(u) > r$ for every real u greater than b. The specification is illustrated as follows:



If f is assumed to be monotonically increasing, the input condition can be simplified. But we do not need to assume that f is increasing or even continuous; if f is not continuous, an exact solution to the equation $f(a) = r$ need not exist, but an exact solution is not required by the specification.

The program we obtain is

$$solve(r, \epsilon) \Leftarrow \widetilde{solve}(\epsilon)$$
$$\widetilde{solve}(\tilde{\epsilon}) \Leftarrow \begin{cases} \text{if } b < a + \tilde{\epsilon} \\ \text{then } a \\ \text{else if } f(\widetilde{solve}(2\tilde{\epsilon}) + \tilde{\epsilon}) \leq r \\ \quad \text{then } \widetilde{solve}(2\tilde{\epsilon}) + \tilde{\epsilon} \\ \quad \text{else } \widetilde{solve}(2\tilde{\epsilon}) \end{cases}$$

In the recursive case, in which $a + \tilde{\epsilon} \leq b$, the $\widetilde{solve}$ program is so closely analogous to the previous binary-search programs as to require no further explanation.

In the base case, in which $b < a + \tilde{\epsilon}$, the output can safely be taken to be a. For, by an input condition, we have

$$f(a) \leq r$$

and (by the other input condition, because $b < a + \bar{\epsilon}$)

$$not\,[f(a + \bar{\epsilon}) \leq r].$$

Thus, a satisfies both conjuncts of the output condition for $\widetilde{solve}$ in this case.

The above program may be regarded as a schema, since we may take the symbol f to be any primitive function symbol. An even more general binary-search program schema can be derived from the specification

$$\begin{aligned} search(r, \epsilon) \;\Leftarrow\; & \text{find } z \text{ such that} \\ & \quad p(r, z) \;\; and \;\; not\, p(r, z + \epsilon) \\ \text{where } p(r, a) \;\; and \;\; & \begin{bmatrix} if\; b < u \\ then\;\; not\, p(r, u) \end{bmatrix} \;\; and \;\; 0 < \epsilon, \end{aligned}$$

where p is a primitive relation symbol and a and b are primitive constants. We obtain the schema

$$search(r, \epsilon) \;\Leftarrow\; \widetilde{search}(\epsilon)$$

$$\widetilde{search}(\bar{\epsilon}) \;\Leftarrow\; \begin{cases} if\; b < a + \bar{\epsilon} \\ then\;\; a \\ else\;\; if\; p(r, \widetilde{search}(2\bar{\epsilon}) + \bar{\epsilon}) \\ \qquad then\;\; \widetilde{search}(2\bar{\epsilon}) + \bar{\epsilon} \\ \qquad else\;\; \widetilde{search}(2\bar{\epsilon}) \end{cases}$$

INTEGER ALGORITHMS

The programs we have discussed apply to the nonnegative real numbers; using the same approach, we have derived analogous programs that apply to the nonnegative integers.

Integer square root

The integer square-root program is intended to find the integer part of $\sqrt{n}$, the real square root of a nonnegative integer n. It can be specified in the theory of nonnegative integers as follows:

$$\begin{aligned} isqrt(n) \;\Leftarrow\; & \text{find } z \text{ such that} \\ & \quad z^2 \leq n \;\; and \;\; not[(z + 1)^2 \leq n]. \end{aligned}$$

In other words, the program must yield a nonnegative integer z that is within 1 less than $\sqrt{n}$.

In the course of the derivation, we are led to introduce an auxiliary program to meet the more general specification

$$\widetilde{isqrt}(\tilde{i}) \Leftarrow \text{find } \tilde{z} \text{ such that } \tilde{z}^2 \leq n \text{ and not } [(\tilde{z}+\tilde{i})^2 \leq n] \quad \text{where } 0 < \tilde{i}.$$

In other words, we wish to find a nonnegative integer $\tilde{z}$ that is within $\tilde{i}$ less than $\sqrt{n}$. This auxiliary specification is precisely analogous to the specification for the real-number square-root auxiliary $\widetilde{sqrt}(\tilde{\epsilon})$, with $\tilde{i}$ playing the role of the error tolerance $\tilde{\epsilon}$.

The motivation for introducing the auxiliary is as follows. In the derivation of the main program $isqrt(n)$, we have the initial goal

	$z^2 \leq n$ *and not* $[(z+1)^2 \leq n]$	z

By resolving this goal with itself and transforming, we obtain the new goal

	$\hat{z}^2 \leq n$ *and not* $[(\hat{z}+2)^2 \leq n]$	*if* $(\hat{z}+1)^2 \leq n$ *then* $\hat{z}+1$ *else* $\hat{z}$

This subgoal is a replica of the original goal, obtained by replacing the term 1 with 2 and the variable z with $\hat{z}$. This suggests introducing the new auxiliary $\widetilde{isqrt}(\tilde{i})$, whose parameter $\tilde{i}$ takes the place of the replaced term 1 in the initial goal. The input condition $0 < \tilde{i}$ for the auxiliary is introduced incrementally, while the derivation of $isqrt(\tilde{i})$ is in progress.

The programs we obtain to meet these specifications are

$$isqrt(n) \Leftarrow \widetilde{isqrt}(1)$$

$$\widetilde{isqrt}(\tilde{i}) \Leftarrow \begin{cases} \textit{if } n < \tilde{i} \\ \textit{then } 0 \\ \textit{else if } \left[\widetilde{isqrt}(2\tilde{i}) + \tilde{i}\right]^2 \leq n \\ \qquad \textit{then } \widetilde{isqrt}(2\tilde{i}) + \tilde{i} \\ \qquad \textit{else } \widetilde{isqrt}(2\tilde{i}) \end{cases}$$

Integer quotient

The integer quotient program can be specified similarly:

$$quot(m, n) \Leftarrow \text{find } z \text{ such that } z \cdot n \leq m \text{ and not } [(z+1) \cdot n \leq m] \quad \text{where } 0 < n.$$

In other words, we wish to find a nonnegative integer z that is within 1 less than m/n, the real-number quotient of m and n.

In the course of the derivation, we are led to introduce an auxiliary subprogram to meet the more general specification

$$\begin{aligned}\widetilde{quot}(\tilde{i}) \;\Leftarrow\; & \text{find } \tilde{z} \text{ such that} \\ & \tilde{z} \cdot n \le m \;\; \textit{and} \;\; \textit{not} \; [(\tilde{z} + \tilde{i}) \cdot n \le m] \\ & \text{where } 0 < n \;\; \textit{and} \;\; 0 < \tilde{i}.\end{aligned}$$

In other words, we wish to find a nonnegative integer $\tilde{z}$ that is within $\tilde{i}$ less than m/n.

The programs obtained to meet these specifications are

$$quot(m, n) \;\Leftarrow\; \widetilde{quot}(1)$$

$$\widetilde{quot}(\tilde{i}) \;\Leftarrow\; \begin{cases} \textit{if } m < \tilde{i} \cdot n \\ \textit{then } 0 \\ \textit{else if } [\widetilde{quot}(2\tilde{i}) + \tilde{i}] \cdot n \le m \\ \qquad \textit{then } \widetilde{quot}(2\tilde{i}) + \tilde{i} \\ \qquad \textit{else } \widetilde{quot}(2\tilde{i}) \end{cases}$$

Here too the derivation is analogous.

THE *LAMBO* FUNCTION

The function *lambo* is a nonnegative-integer approximation for the inverse of a given nonnegative integer function f. We assume that f has the following properties:

f is monotonically increasing, i.e.,

$$\begin{aligned}&\textit{if } u \le v \\ &\textit{then } f(u) \le f(v),\end{aligned}$$

f is unbounded, i.e.,

$$(\exists h)[u \le f(h)],$$

for all nonnegative integers u and v. Here h also ranges over the nonnegative integers.

The specification for the desired program is

$$\begin{aligned}lambo(n) \;\Leftarrow\; & \text{find } z \text{ such that} \\ & n \le f(z) \;\; \textit{and} \\ & (\forall g)[\textit{if } g < z \;\; \textit{then } \; f(g) < n].\end{aligned}$$

In other words, *lambo*(n) is the least nonnegative integer z such that $n \leq f(z)$. Note that this specification depends on the given function f: for a different function f, we obtain a different specification and, presumably, a different program.

A linear-time *lambo* program was derived by Dijkstra [82], who used transformations of that program to provide a novel proof of a theorem of Lambek and Moser—hence the name of the function. The derivation of a *lambo* program was posed as an exercise for participants at the 1985 Workshop on the Specification and Derivation of Programs, in Marstrand, Sweden. A construction analogous to our square-root derivation turns out to yield a binary-search *lambo* program. We outline that derivation briskly here.

We begin with the tableau

assertions	goals	outputs *lambo*(n)
	1. $n \leq f(z)$ *and* *if* $g(z) < z$ *then* $f(g(z)) < n$	z

Here $g(z)$ is a skolem function obtained by eliminating the quantifier $(\forall g)$ from the specification. The unboundedness of f is expressed by the assertion

2. $u \leq f(h(u))$		

where h is a skolem function introduced to eliminate the quantifier $(\exists h)$. (Note that, by duality, existential quantifiers in assertions are treated in the same way as universal quantifiers in goals.) The monotonicity of f is not represented by an assertion; it is declared, and treated by the special-relation rules.

Using the property of the nonnegative integers

$$not\,(u < 0),$$

taking z to be 0, we reduce our initial goal 1 to

	3. $n \leq f(0)$	0

In other words, in the case in which $n \leq f(0)$, our original goal is true and the output 0 meets the specification.

Returning to our initial goal 1, using the property

$$u < v + 1 \;\equiv\; u \leq v,$$

we can develop the goal

	4. $f(z') < n$ *and* $n \leq f(z'+1)$	$z'+1$

From an intuitive standpoint, if this goal is true for some z', the original goal 1 is true, taking z to be $z'+1$. For then $n \leq f(z'+1)$ and, if we assume that $g(z'+1) < z'+1$, we have $g(z'+1) \leq z'$ (by the property), hence $f(g(z'+1)) \leq f(z')$ (by monotonicity), and hence $f(g(z'+1)) < n$ (by goal 4 and transitivity). Thus, both conjuncts of the initial goal 1 are true. In the system, the goal is obtained by a special-relation rule.

Goal 4 is analogous to the initial goals of our other derivations. Theory resolution of the goal with itself, invoking the property

$$u \leq v \ \ or \ \ v < u,$$

yields the new goal (after transformation)

	5. $f(\hat{z}) < n$ *and* $n \leq f(\hat{z}+2)$	*if* $f(\hat{z}+1) < n$ *then* $\hat{z}+2$ *else* $\hat{z}+1$

This is a replica of our previous goal 4, obtained by replacing the constant 1 with the constant 2. This suggests forming an auxiliary subprogram, which we shall call $limbo(\tilde{i})$, with output condition

$$f(\tilde{z}) < n \ \ and \ \ n \leq f(\tilde{z}+\tilde{i}).$$

Two input conditions,

$$0 < \tilde{i} \quad \text{and} \quad f(0) < n,$$

are introduced incrementally during the derivation of *limbo*. In short, the ultimate specification for the subprogram is

$$\begin{aligned} limbo(\tilde{i}) \ \Leftarrow \ & \text{find } \tilde{z} \text{ such that} \\ & f(\tilde{z}) < n \ \ and \ \ n \leq f(\tilde{z}+\tilde{i}) \\ & \text{where } 0 < \tilde{i} \ \ and \ \ f(0) < n. \end{aligned}$$

An assertion describing the auxiliary *limbo* is introduced into the main tableau; we can then complete the main derivation, obtaining the program

$$lambo(n) \ \Leftarrow \ \begin{cases} if \ n \leq f(0) \\ then \ 0 \\ else \ limbo(1)+1. \end{cases}$$

The derivation of the auxiliary *limbo* closely resembles the other binary-search derivations. We obtain the program

$$limbo(\tilde{i}) \Leftarrow \begin{cases} \textit{if } n \leq f(\tilde{i}) \\ \textit{then } 0 \\ \textit{else if } f(limbo(2\tilde{i}) + \tilde{i}) < n \\ \quad \textit{then } limbo(2\tilde{i}) + \tilde{i} \\ \quad \textit{else } limbo(2\tilde{i}) \end{cases}$$

(As usual, the three recursive calls can be combined by common-subexpression elimination and the program can be transformed into an iterative equivalent.)

The well-founded relation that serves as the basis for the induction (and the termination argument) is again the bounded-doubling relation $\prec_{bd(y)}$. The upper bound y in this case is $h(n)$, where h is the skolem function in the unboundedness assertion

$$u \leq f(h(u)).$$

(Therefore, $h(n)$ is an argument that will force f to exceed the given integer n.) For, intuitively speaking, if the parameter $\tilde{i}$ of *limbo* exceeds this upper bound, that is, if

$$h(n) < \tilde{i},$$

we have

$$f(h(n)) \leq f(\tilde{i})$$

(by the monotonicity of f) and hence

$$n \leq f(\tilde{i})$$

(by the unboundedness assertion and transitivity). In this case, the *limbo* program exits via the base case; the recursive call is not executed. Consequently, the upper bound on $\tilde{i}$ is maintained whenever the recursive call is executed, and termination is not endangered. In the derivation, of course, this argument is conducted within the rules of the system.

Note that, in this example, the choice of the well-founded relation $\prec_{bd(h(n))}$ depended on the skolem function h. This function is not primitive; we are told that an argument exists that will cause f to exceed the given integer, but we are not told how to compute such an argument. For this reason, the *lambo* example has sometimes been regarded as a challenge to systems that extract programs from purely constructive mathematical proofs (e.g., Martin-Löf [79], Sato [79], Nordström and Smith [84], Bates and Constable [85]). In such a system, a quantity exists only if we have the means to compute it. Here we deal with a quantity that, we are told, exists — but we have no means to compute it; however, we do not need such a computation, because the quantity's precise value has no bearing on the output.

9. CONCLUSION

The examples in this paper serve to illustrate the application of the deductive-tableau system. In a more general sense, they suggest ways in which a mechanical system might invent a novel programming concept.

The results of this investigation run counter to our usual experience. It is common for a bit of apparently simple and intuitively straightforward reasoning to turn out to be difficult to formalize and even more difficult to duplicate automatically. Here the opposite is true: an idea that requires a substantial leap of human ingenuity to discover is captured in a few easy formal steps. We may consequently imagine that truly original ideas will arise from the fortunate application of simple mechanisms.

ACKNOWLEDGMENTS

We would like to thank Martin Abadi, Michael Beeson, Ron Burback, Bengt Jonsson, Yoni Malachi, Eric Muller, Larry Paulson, Mark Stickel, and Jonathan Traugott for helpful discussions and constructive suggestions on the subject of this paper; and Evelyn Eldridge-Diaz, for TEXing many versions of the manuscript. The square-root derivation was reproduced with an interactive program-synthesis system by Frank Yellin.

REFERENCES

Bates and Constable [85]
J. L. Bates and R. L. Constable, Proofs as programs, *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 1, January 1985, pp. 113–136.

Dershowitz and Manna [77]
N. Dershowitz and Z. Manna, The evolution of programs: Automatic program modification, *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 6, November 1977, pp. 377–385.

Dijkstra [82]
E. W. Dijkstra, Lambek and Moser revisited (Function property proving), in *Theoretical Foundations of Programming Methodology* (M. Broy and G. Schmidt, editors), Reidel, Dordrecht, Holland, 1982, pp. 19–23.

Harrison and Khoshnevisan [86]
P. Harrison and H. Khoshnevisan, Efficient compilation of linear recursive functions into object-level loops, *SIGPLAN '86 Symposium on Compiler Construction*, Palo Alto, Calif., June 1986, pp. 207–218.

Manna and Waldinger [80]
Z. Manna and R. Waldinger, A deductive approach to program synthesis, *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 1, January 1980, pp. 90–121.

Manna and Waldinger [86]
Z. Manna and R. Waldinger, Special relations in automated deduction, *Journal of the ACM*, Vol. 33, No. 1, January 1986, pp. 1–59.

Martin-Löf [79]
P. Martin-Löf, Constructive mathematics and computer programming, *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, Hannover, August 1979.

Murray [82]
N. V. Murray, Completely nonclausal theorem proving, *Artificial Intelligence*, Vol. 18, No. 1, 1982, pp. 67–85.

Nordström and Smith [84]
B. Nordström and J. Smith, Propositions and specifications of programs in Martin-Löf's type theory, *BIT*, Vol. 24, 1984, pp. 288–301.

Robinson [65]
J. A. Robinson, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, Vol. 12, No. 1, January 1965, pp. 23–41.

Robinson [79]
J. A. Robinson, *Logic: Form and Function*, North-Holland, New York, N. Y., 1979.

Sato [79]
M. Sato, Towards a mathematical theory of program synthesis, *Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan, August 1979, pp. 757–762.

Smith [85]
D. R. Smith, Top-down synthesis of simple divide-and-conquer algorithms, *Artificial Intelligence*, Vol. 27, No. 1, September 1985, pp. 43–96.

Stickel [81]
M. E. Stickel, A unification algorithm for associative-commutative functions, *Journal of the ACM*, Vol. 28, No. 3, July 1981, pp. 423–434.

Stickel [85]
M. E. Stickel, Automated deduction by theory resolution, *Journal of Automated Reasoning*, Vol. 1, No. 4, 1985, pp. 333–355.

Traugott [86]

J. Traugott, Deductive synthesis of sorting programs, *Eighth International Conference on Automated Deduction*, Oxford, England, July 1986, pp. 641–660.

Wensley [59]

J. H. Wensley, A class of nonanalytical iterative processes, *Computer Journal*, Vol. 1, January 1959, pp. 163–167.

END

DATE

FILMED

DTIC

JULY 88